THESIS

RADIATION TESTING OF THE CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP) FOR SPACE-BASED APPLICATIONS

by

James C. Coudeyras

December 2005

Thesis Co-Advisors:	Herschel H. Loomis, Jr. Alan A. Ross

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Radiation Testing of the Configurable Fault Tolerant Processor (CFTP) for Space-Based Applications	**5. FUNDING NUMBERS**
6. AUTHOR(S) James Coudeyras	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Field Programmable Gate Arrays (FPGAs) provide a reconfigurable asset in the design of space computing. "Hardware" configurations are stored in FPGA memory elements, which are susceptible to Single Event Upsets (SEUs). What is the best way to detect and mitigate SEUs and correct them before they become functional errors? The Configurable Fault Tolerant Processor (CFTP) consists of a controller FPGA (X1) controlling an experiment FPGA (X2), which can be used to test different fault-mitigation techniques. This focus of this thesis was to develop and execute a radiation test plan to evaluate different experiments in a proton radiation beam at Crocker Nuclear Laboratory, Davis, CA. A shift register was designed to determine a proton flux conducive to SEU observation. The shift register was also modified to create two additional configurations, implemented with the memory elements of the Look-Up Table and Flip-flops within an FPGA Configurable Logic Block. The data collected from this program was then analyzed for SEU rates and fault susceptibility. This data was extrapolated using a radiation environment model to predict the on-orbit SEU-rate for CFTP in the NPSAT1 orbit of 560 km, 35.4 degrees inclination, as well as Virtex II FPGAs and at 1000 and 1500 km altitudes.

14. SUBJECT TERMS Field Programmable Gate Array (FPGA), FPGA testing, FPGA radiation testing, Single Event Effect (SEE), Single Event Upset (SEU), SEU prediction.	**15. NUMBER OF PAGES** 165
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

RADIATION TESTING OF THE CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP) FOR SPACE-BASED APPLICATIONS

James C. Coudeyras
Lieutenant Commander, United States Navy
B.S., University of Kansas, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2005

Author:	James C. Coudeyras
Approved by:	Herschel H. Loomis, Jr. Thesis Co-Advisor
	Alan A. Ross Thesis Co-Advisor
	Jeffrey B. Knorr Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Field Programmable Gate Arrays (FPGAs) provide a reconfigurable asset in the design of space computing. "Hardware" configurations are stored in FPGA memory elements, which are susceptible to Single Event Upsets (SEUs). What is the best way to detect and mitigate SEUs and correct them before they become functional errors? The Configurable Fault Tolerant Processor (CFTP) consists of a controller FPGA (X1) controlling an experiment FPGA (X2), which can be used to test different fault-mitigation techniques. This focus of this thesis was to develop and execute a radiation test plan to evaluate different experiments in a proton radiation beam at Crocker Nuclear Laboratory, Davis, CA. A shift register was designed to determine a proton flux conducive to SEU observation. The shift register was also modified to create two additional configurations, implemented with the memory elements of the Look-Up Table and Flip-flops within an FPGA Configurable Logic Block. The data collected from this program was then analyzed for SEU rates and fault susceptibility. This data was extrapolated using a radiation environment model to predict the on-orbit SEU-rate for CFTP in the NPSAT1 orbit of 560 km, 35.4 degrees inclination, as well as Virtex II FPGAs and at 1000 and 1500 km altitudes.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

GLOSSARY

ACS	Attitude Control Subsystem
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
C&DH	Command and Data Handler
CFTP	Configurable Fault Tolerant Processor
CLB	Configuration Logic Block
CORDIC	COordinate Rotation DIgital Computer
EDU	Engineering Development Unit
EMC	Electromagnetic Compatibility
FET	Field Effect Transistor
FPGA	Field Programmable Gate Array
I/O	Input/Output
LEO	Low Earth Orbit
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MidSTAR1	Midshipman Space Technology Applications Research Satellite mod 1
MIL-STD	Military Specification Standard
MJA	Major Address
NPS	Naval Postgraduate School
NPSAT1	Naval Postgraduate School Spacecraft Architecture and Technology Demonstration Satellite
RF	Radio Frequency
SEE	Single Event Effect
SEU	Single Event Upset
SPC	Statistical Process Control
STP	Space Test Program
TCV	Technology Characterization Vehicle
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy

VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

ACKNOWLEDGMENTS

I would first like to thank my wife, Melissa, and my boys, Alex and Austin. They have been extremely patient and understanding with my desire to “expand my horizons” and earn a degree in Electrical Engineering, when earning an Astronautical Engineering degree would have been the “easier” route. Thanks for making me the luckiest man around.

To my advisors and CFTP project mates: Thanks for everyone’s help and devotion. Many long hours were put into this project, not only for my thesis, but every aspect of making CFTP work. I couldn’t have done my part without a lot of groundwork before me. More importantly, thanks for making a tedious and trying process so enjoyable. I promised myself I would only stay in the Navy as long as I was having fun. If I’m working with people like you, I’ll be in the Navy for a long time to come.

To my classmates: This has truly been the best group I’ve had the pleasure to work with. It’s rare to find a group so large that works so well together. Fair winds and following seas (that means best of luck, Super Dave) at your next assignment.

To my professors: Thanks for opening my eyes and helping me develop an understanding of material that eluded my grasp in my undergraduate days.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Computing power in space has been limited in the past by the need to qualify a design early in the acquisition process, in order to ensure that the processor will operate properly in the space environment for the lifetime of the satellite. Field Programmable Gate Arrays (FPGAs) offer the advantage of reconfigurable computing, where the “hardware” configuration is stored in memory elements in an FPGA. These memory elements are susceptible to Single Event Upsets (SEUs). SEUs, caused by the radiation environment in space, can have varying effects on a spacecraft ranging from no functional effect to rendering the spacecraft useless. These SEUs must be detected and their effects mitigated to best utilize the advantages of using FPGAs.

Continuous testing is done to verify best design practices for configuration and data redundancy in FPGA design. The Configurable Fault Tolerant Processor (CFTP) provides a test platform to validate different designs, tools, and design techniques. This thesis describes the development and testing process of the CFTP and a brief overview of testing for space applications, using the testing of the CFTP in preparation for the launch environment as an example. The focus of this thesis is the development and execution of a radiation test plan for use at the Crocker Nuclear Laboratory in Davis, CA. A shift register was designed as a test circuit to determine a proton-flux level that was conducive to SEU observation, while running experiments. The test plan describes two variants of the CFTP that were tested. The first CFTP design, similar to the flight design for Naval Postgraduate School Spacecraft Architecture and Technology Demonstration Satellite (NPSAT1), uses a Xilinx Virtex I (600,000-gate-equivalient) FPGA as the experimental FPGA. The second CFTP design uses a Xilinx Virtex II (6-million-gate-equivalent) FPGA as the experimental FPGA.

The test plan provides a list of test equipment and a diagram of the test layout that was used in Davis, CA. Three programs were tested in the proton radiation beam: a shift register, a CORDIC algorithm, and PIX (a distributed triple-modular redundant, MIPS-based processor). The shift register was used to maximize coverage and the probability

of detecting an SEU when one occurs in the proton beam. These results were used to establish a proton flux to yield an SEU every 10-30 seconds and extrapolated to establish a baseline for SEU prediction on orbit.

This shift register was also modified to create two additional configurations by utilizing different combinations of the SRL16 macro, implemented with the memory elements of the Look-Up Table (LUT), and Flip-flops within an FPGA Configurable Logic Block (CLB). A COordinate Rotation DIgital Computer (CORDIC) algorithm was also tested on both CFTP boards to validate the benefit to using Triple Modular Redundancy (TMR) and partial reconfiguration.

Focus was then shifted to testing the PIX processor, a distributed-TMR, MIPS-based processor, designed by Majewicz [7]. The design is too large to fit on the Virtex I-CFTP, so it was tested solely on the Virtex II-CFTP.

Prior to radiation testing, two fault injection techniques, using FPGA Editor (Xilinx ISE software package) and JBits 2.8 (Xilinx), were used to verify proper operation of the experments and determine expected output during testing. Proton radiation testing of fault mitigation designs occurred in the cyclotron at University of California-Davis, and results follow the fault injection section. The data collected from the shift register program was then analyzed for SEU rates and fault susceptibility. This data was extrapolated using CREME96 to predict the on-orbit SEU-rate for CFTP in the NPSAT1 orbit of 560 km, 35.4 degrees inclination, as well as Virtex II FPGAs and at 1000 and 1500 km altitudes.

These results validated the fault-injection tool and were extrapolated to provide an estimate of the on-orbit SEU-rate for NPSAT1 as 1 SEU every 6-7 days. Radiation test results showed a relatively consistent SEU-rate for different programs, as was expected. This thesis has only scraped the surface for data collection and analysis for CFTP. Numerous opportunities for future research exist and are required for a more complete guide to the performance of CFTP.

Current programs should be run with the fault injection tool for longer periods to approach asymptotic values (small variances between tests.) New algorithms need to be developed and tested to explore better fault mitigation techniques. TMR methods were tested with the CORDIC, and PIX implements a distributed TMR design. Quadded-logic methods could also be tested.

PIX is too large to fit on CFTP-1. Research should be completed to reduce the size of PIX and test this smaller design on CFTP-1. Finally, an experiment agenda needs to be developed for the utilization of CFTP aboard NPSAT1 and MidSTAR1.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Computing power in space has been limited in the past by the need to qualify a design early in the acquisition process, in order to ensure that the processor will operate properly in the space environment for the lifetime of the satellite. As today's information technology continues to expand according to Moore's law [1], spacecraft designers are limited from using the most current technology. Additionally, hardware designs must be completed, tested, and software written for vital spacecraft functions within the design/acquisition timeline. This problem is mitigated to a degree by the use of Field Programmable Gate Arrays (FPGAs). A "hardware" configuration is stored in memory elements in an FPGA, and FPGAs are reconfigurable. This allows for testing and qualifying the actual hardware, but also allows for continued development of the application configuration that will be instantiated on the FPGA. These save time and money in the design of systems, as well as redesign costs to fix errors. The trade space with FPGAs, though, is reliable computing.

With the increased usage of FPGAs in space applications, reliable computing is a continuing area of concern. Single event effects (SEEs), caused by the radiation environment in space, can have varying effects on a spacecraft ranging from no functional effect to rendering the spacecraft useless. These SEEs can be mitigated with good designs that use redundancy techniques for data verification. With the advent of FPGAs, the "hardware" now becomes susceptible to SEEs, where previous hardware configurations were not. In an FPGA, the configuration is instantiated into a series of logic blocks consisting of Look-Up Tables, Flip-Flops, and control signals and signal routing [2]. Because this configuration is not hardware but memory contents, an upset bit is no longer necessarily just a data error, but could result in a configuration error. In a worst-case scenario, this could lead to a new configuration that disables the spacecraft permanently.

To avoid such drastic consequences, continuous testing is done to verify best design practices for configuration and data redundancy in FPGA design. The Configurable Fault Tolerant Processor (CFTP) provides a test platform to validate different designs, tools, and design techniques. This thesis reports on the testing of some of those designs in the cyclotron at University of California-Davis, as well as the application of the JBits program by Xilinx, Corp to the determination of fault tolerance [3]. Test results also establish a baseline for on-orbit operation for CFTP, which will be flying on the Naval Postgraduate School Spacecraft Architecture and Technology Demonstration Satellite (NPSAT1) and the Midshipman Space Technology and Applications Research Satellite mod 1 (MidSTAR). Both satellites will be launched on an Atlas V-401 in October 2006 as part of the Space Test Program (STP-1), shown in Figure 1.

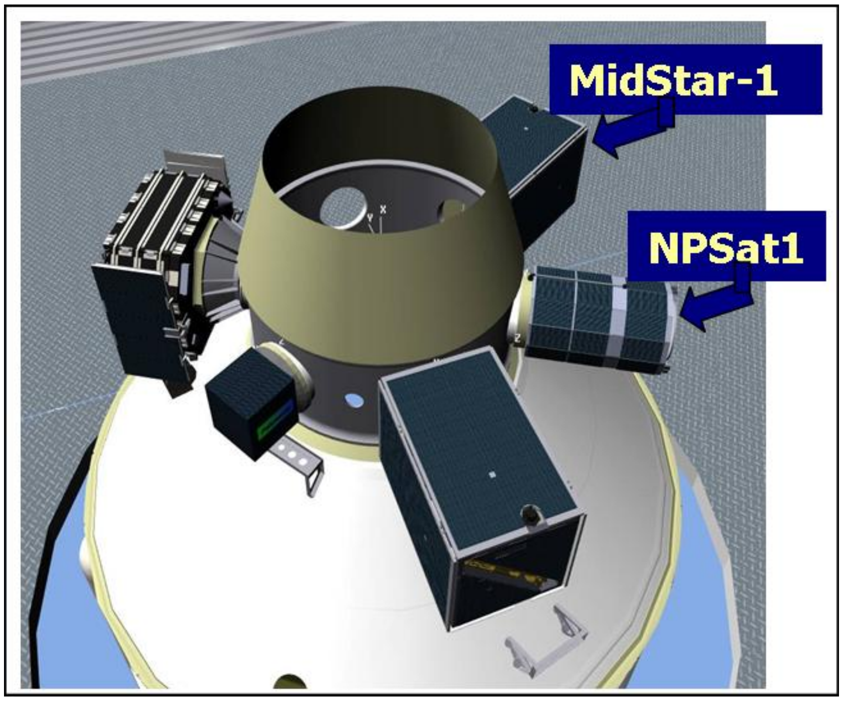


Figure 1. STP-1 Payloads (from [13])

Chapter II provides a brief summary of the space radiation environment and the design of CFTP with appropriate references cited for a more detailed description. Following the background of the intended operating environment and processor design, a review of various testing methods is provided in Chapter III. Radiation testing is a small portion of the testing that is completed for any electronic device, especially one intended for space applications.

For a complete understanding of testing involved with electronics, hardware validation and software testing are first described, leading into integration testing, where the CFTP design is validated. Because CFTP is designed to operate in space, a review of the space qualification process and associated environmental testing is then provided. One part of environmental testing is radiation testing. Radiation in space (high-energy particles) contributes to two primary effects on electronics. The first is deterioration of the electronics due to Total Ionizing Dose (TID), and the second is Single Event Effects (SEE) [4]. Chapter IV describes radiation test preparations and test design requirements to address the second effect, of which the primary concern is Single Event Upsets (SEUs).

Three test circuits were developed for this phase of the development. The first test circuit is a series of shift registers instantiated on X2 (the experiment FPGA) to utilize as much area as possible. This was developed to maximize coverage and the probability of generating an error when an SEU occurs in the proton beam. A maximal-length Linear Feedback Shift Register (LFSR) was used on X1 (control FPGA) to generate pseudo-random 1s and 0s. These data bits were sent to X2 for shifting through the registers. The output of each series of shift registers was then processed to identify potential data-bit SEUs. This design was tested on a Virtex-1 board (CFTP-1) and a similar board that utilizes a Virtex-II FPGA as X2 (CFTP-2) [5]. These results were extrapolated to establish a baseline for SEU prediction on orbit.

A COordinate Rotation DIgital Computer (CORDIC) algorithm was also tested on both CFTP boards to validate the benefit to using Triple Modular Redundancy (TMR) and partial reconfiguration. An alternate version of the CORDIC that does not use TMR was also tested [6].

Focus was then shifted to testing the PIX processor designed by Majewicz [7]. This is a MIPS 3000 processor that uses a distributed, pipelined, TMR design. The design is too large to fit on CFTP-1, so it was only tested on CFTP-2.

All three programs were first tested with fault-simulation techniques. Xilinx ISE software and JBits were used to emulate faults prior to testing at UC-Davis. These fault simulation techniques and the testing results are described in Chapter V. These results were then compared to results from radiation testing. Test results validated operation of these programs in a radiation beam, as well as validating the fault-simulation tools. This data also provides a baseline for SEU prediction on orbit and a basis for follow-on programs to be tested with the CFTP experiment.

Finally, Chapter VI summarizes testing results and provides predictions for on-orbit operation. Additionally, recommendations are provided for future radiation testing at UC-Davis and on-orbit experiments. While ground testing cannot perfectly imitate the conditions and actual events in space, radiation testing is an extremely useful tool to validate electronics design and predict system operation on-orbit.

II. BACKGROUND

Electronic components are susceptible to single event effects (SEEs) when operated in space. This is due to the high-radiation environment that consists of high-energy neutrons, protons, and heavy ions [8]. This environment must be understood, and protection against its effects needs to be incorporated into the design of systems intended for space applications. Application Specific Integrated Circuits (ASICs) have been designed to survive this environment through radiation hardening and redundancy. The hardware configuration of these devices is not susceptible to bit upsets. With the advent of Field Programmable Gate Arrays (FPGAs), the design configurations are stored in memory as configuration information. A primary advantage is the flexibility to change design errors through modification of the configuration information, or even upload a new configuration to accomplish a new mission. Software re-configuration capability poses a problem in space, though.

In ASICs, SEEs might change a data bit, which could be restored through good design and incorporation of error detection and correction (EDAC) circuitry. With FPGAs, an SEE could change the configuration of the processor and render the equipment or even the spacecraft useless. Due the extensive resource investment (time and money) into each spacecraft, the risk of losing the function of a satellite due to SEE is unacceptable [9]. Techniques must be incorporated to mitigate the adverse effects of SEEs, more specifically Single Event Upsets (SEUs). One such approach is to use triple modular redundancy (TMR) with partial reconfiguration [10]. This approach is at the core of the design for the Configurable Fault Tolerant Processor (CFTP). CFTP utilizes two FPGAs, one as a controller and the other as an experiment platform. The TMR processor is instantiated on the experiment FPGA, where the data is majority voted. Ideally, in the event of an SEU, the corrupted data is voted out. In his thesis, Hulme provided a thorough explanation of the CFTP design and mitigation options and selection [8].

CFTP is also designed to provide a testbed for other experiments. Various methods of testing these designs exist, but this thesis focuses on three areas:

configuration alteration using Xilinx FPGA Editor, fault injection with a JBits-based tool, and radiation testing [11]. Radiation testing used proton radiation produced at the cyclotron of the Crocker Nuclear Laboratory at the University of California-Davis. To better understand the results of radiation testing, a background of the design, assembly, and testing of CFTP is provided in the next chapter.

III. INTEGRATION AND TESTING

A. TYPES OF TESTING

Descriptions of testing for computers designed for the space environment begin with two branches: hardware and software. Both items are tested at the unit level up through the system level. After software is merged with hardware, testing proceeds to verify proper integrated system operation. Testing then continues for on-orbit calibration, verification of proper equipment operation and potential for troubleshooting [9].

1. Hardware Testing

Hardware testing starts with component testing. The process then proceeds through acceptance testing to a functional test. The following describes the hardware manufacturing and test process used by David Rigmaiden, Electronics Technician for the NPS Space Systems Academic Group, for CFTP [12].

Component testing started with part selection. Because CFTP was designed to operate in the space environment, radiation hardened parts were desired. CFTP is an experiment, and as such, is not strictly limited to only radiation-hardened components for production. The CFTP flight board will instead consist of military specification (MILSPEC) parts, with the exception of the power supplies, which are one grade higher than MILSPEC, and the FPGAs, which are radiation hardened. After the components were received and the design board had been fabricated, a fit test was performed. The assembled board is pictured in Figure 2. This test ensured that all components fit on the board as designed. If components do not integrate as designed, further investigation is required to determine whether a part is faulty, if the circuit board was improperly fabricated, or if the design is bad. After the fit test yields the desired results, the board is then assembled without major components. This consists of the voltage regulators and passive components, such as resistors and capacitors [12].

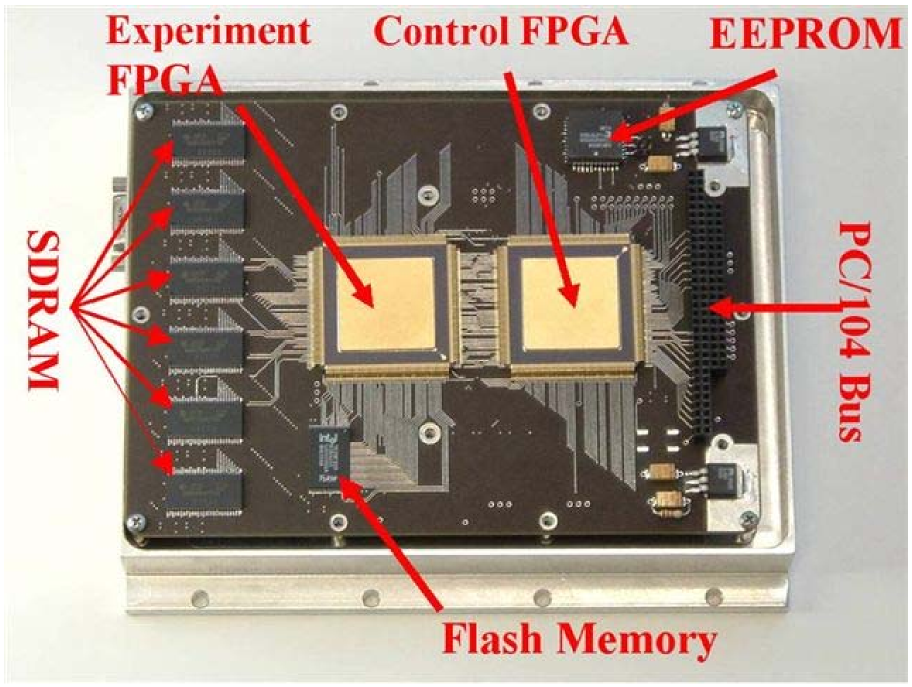


Figure 2. Assembled CFTP Board (from [13])

Power is then supplied to the board with the base power supply, which is a 5-volt source for CFTP. All pins are tested for the expected voltage, and the power is left on for at least one day to verify the board is capable of sustaining a continuous load. Power is removed from the board, and the simpler silicon devices, consisting of the Flash memory, Synchronous Dynamic Random Access Memory (SDRAM), and Electrically Erasable Programmable Read Only Memory (EEPROM), are attached. The board is then powered up again, and the board is checked to verify serial communication is capable with the EEPROM [12]. The boot program and test bench for the EEPROM was generated using Xilinx ISE software.

The configuration control FPGA (X1) is then attached to the board with a thermal epoxy and soldered at all pin connections with additional solder at the corners. The solder at the corners, termed “spiking,” provides additional stiffness. The epoxy serves two purposes. The first is to provide additional support to the solder joints during the high vibration environment experienced during launch. The second purpose is to provide a heat sink for the FPGA during operation, which helps provide heat dissipation in addition to the heat radiated from the exposed face of the FPGA [12].

After the solder and epoxy have hardened, a clock signal is sent from the boot EEPROM out through X1 to verify the pin connections. After a clock signal is verified, the experiment FPGA (X2) is soldered and glued to the circuit board, and the board is ready for software integration testing [12].

2. Software Testing

A primary method of software testing is accomplished with simulation. For example, the CFTP design uses a Virtex I Pro 600 (XQV600) FPGA for both the configuration/experiment control processor (X1) and the experiment processor (X2). The programming for X1 has multiple modules that comprise the whole, all of which must be tested. The Xilinx Integrated Software Environment (ISE) software can be used to instantiate these modules, written in VHDL, on a given FPGA (XQV600 in this case). The ISE software enables users to "synthesize and compile FPGA and Complex Programmable Logic Devices (CPLD) devices" [14]. A testbench can then be designed with ModelSim or other simulation software to verify that the modules and overall design operates as expected. A common and highly recommended practice for any software design and testing is to plan the design in stages and test each stage as it is completed [15]. This accomplishes two things. The first is that this approach is conducive to a modular design, which allows for parallel design/programming and enables easier implementation of upgrades. The second is that any coding errors are located earlier in the design process, saving time and cost.

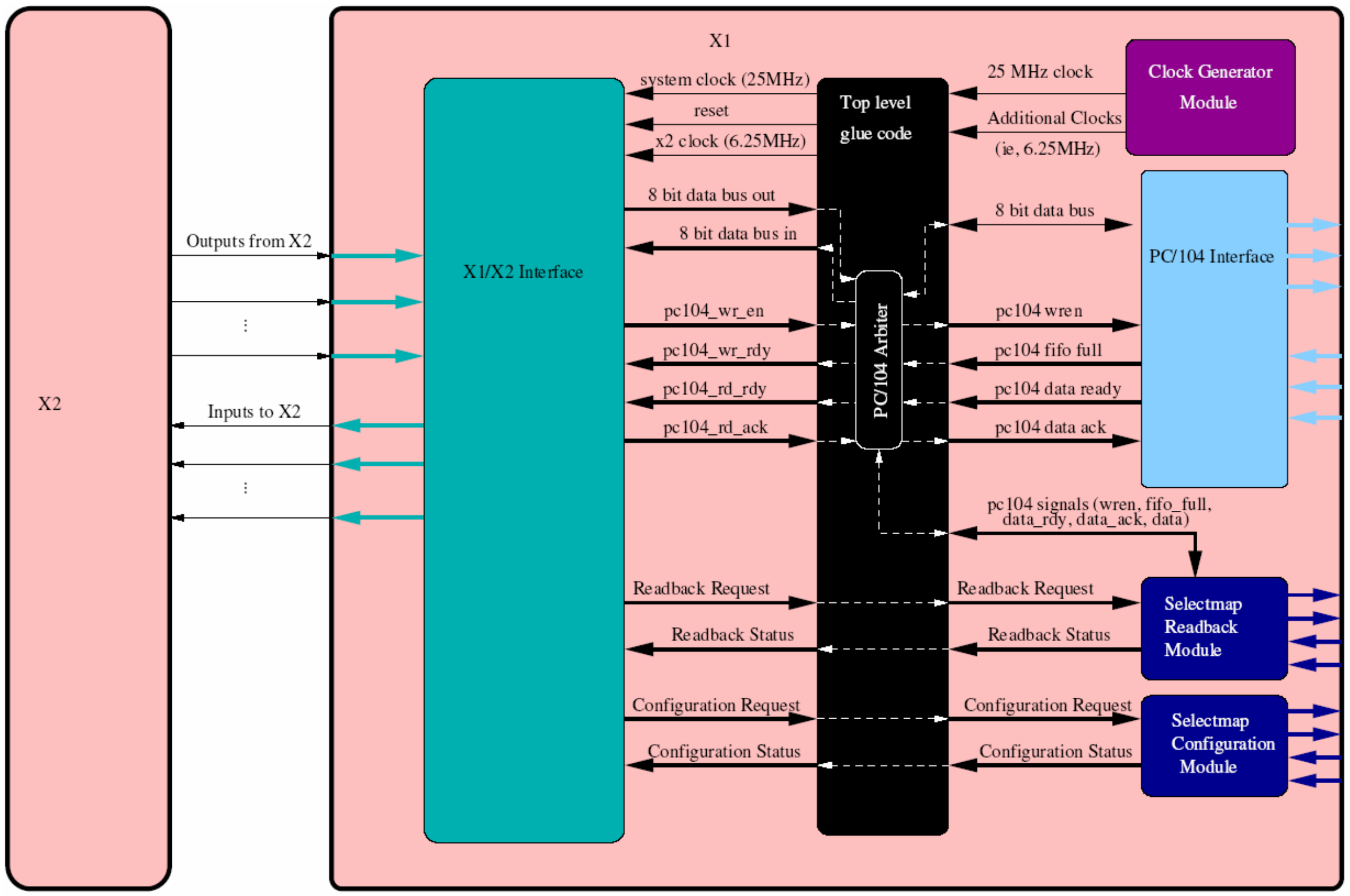
Various simulation programs exist to help verify proper operation. As mentioned in the example above, ModelSim was used in conjunction with ISE for parts of the CFTP testing [16]. ModelSim supports VHDL simulation. Not all simulation can be accomplished with a commercial program, though. A program to simulate the design must be designed when no product is readily available. This was necessary for the configuration control FPGA (X1) of CFTP due to the complexity of the design. These programs and simulations are modified to test a given design.

All operation and interface software was developed by Mindy Surratt, research associate for the CFTP project [17]. A complete picture of the interface and control modules on X1 is shown in Figure 3. The initial step was to develop the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) code to interface from the CFTP board through the PC/104 bus to the ARM processor. (The surrogate processor for the development board is a TMZ104 processor.) The ARM processor is the interface between the CFTP experiment and data bus of the spacecraft (Command and Data Handler (C&DH) for NPSAT-1.) As the PC/104 module was developed and simulated, the CFTP board was completely assembled. The PC/104 module was integrated with and tested on the hardware, and further modules were developed and tested on the cftp board.

3. System Integration and Testing

Testing was successfully accomplished by programming X1, through the JTAG (IEEE 1149.1/Boundary Scan), port using iMPACT (part of ISE) to write a constant value ("2" was used) from X1 to the sTMZ104 through the PC/104 bus. JTAG is a testing standard that "defines a hardware architecture and the mechanisms" to support circuit board testing using software solutions [18]. iMPACT is the Xilinx download tool that uses a parallel connection to the JTAG port on the FPGA [19]. The next step was to verify write from processor board to X1, through the PC/104 bus. X1 was programmed to read back various inputs to verify successful read/write communication via the PC/104. At this point, the VHDL testbenches and ModelSim were used to simulate the PC/104 modules to determine the clocking sequence and the Xilinx Coregen (part of ISE) program was used to create the First In, First Out (FIFO) module to act as a buffer, as well as adding flags for "handshaking" to avoid data conflicts [17].

Communication between X2 and X1 was tested by programming X2 via JTAG to pass a 25 MHz clock to X1. X1 then printed a "2" out to the PC/104 on 1-to-0 transition, indicating a clock signal was present. Additionally, the signal was routed to an Input/Output (I/O) pin, which was connected to an oscilloscope. After X2 was successfully programmed via JTAG, X2 was configured with a 25 MHz clock from X1 through Selectable Microprocessor Access Port (SelectMAP). SelectMAP is the fastest method of programming an FPGA, because it uses a byte-wide bidirectional-access port for reading and writing the configuration data [20]. The 14 I/O-pin and 3 Mode-pin connections are shown in Figure 4.



Readback/config request: high for one clock starts readback/config
Readback/config status: high when readback/config active, low otherwise
pc104_wr_rdy: high when ok to write to pc104
pc104_wr_en: high for one clock writes data to pc104
pc104_rd_rdy: high when data available from pc104, rd_ack high for one clock brings rd_rdy back low

X1 to X2 pinouts (variable number of pins, up to 45)
X1 Selectmap pinouts (12 pins)
X1 PC/104 pinouts (21 pins)
X1 intra–module internal signals

NOTE: Figure not drawn to scale.

If you need additional (slower) clocks, add them to clockGen and pass the output to x2Int

There are also 42 flash pins used in the top level design

Figure 3. X1 Software Modules (from [17])

Partial reconfiguration was then tested by partially reconfiguring X2 to divide the clock down to 12.5 MHz. Now that programming and partial-reconfiguration tests were successful, SelectMAP readback was verified by programming X2 via JTAG, then reading out the configuration to X1 via SelectMAP out to the PC/104.   This concluded in successful functional tests of the communication between X1 and X2 and the PC/104 bus. The software module to enable read and write capability between the Flash memory and X1 was developed next. After successfully erasing, writing, and reading all blocks of the Flash memory, the SelectMap interface between X1 and X2 was developed. Note that the capability to read and write to flash from X2 exists but has not been tested yet.

CFTP utilizes the SelectMAP interface, which provides an 8-bit bidirectional data-bus interface, to the Virtex configuration logic [21], as shown in Figure 4. The SelectMAP communications were tested by first programming a clock on X2 via JTAG and sending the clock signal through X1 out to the PC/104 bus. X2 was then reprogrammed via SelectMAP from X1 to send a different clock signal (the original clock was divided by two). The clock output on the PC/104 connection was also probed with an oscilloscope to verify the clock had the proper frequency. This same process was used to verify proper operation of the SelectMAP readback module. The separate modules were then incorporated into a top level program and integrated with the hardware for further testing.

With all software modules tested and operational, an experiment was chosen to test for proper operation. The process for developing and testing an experiment/algorithm is shown in Figure 5. A COordinate Rotation DIgital Computer (CORDIC) algorithm designed by Josh Snodgrass was chosen, because this algorithm was previously designed to operate on the Xilinx XQVR600 used in CFTP [17]. This algorithm was instantiated on X1 and X2. The algorithm was provided with a sequence of 32-bit 2's complement numbers starting at zero and incremented by a binary digit. This algorithm uses iterative shifts and adds to perform vector rotations of the input angle [22]. The output from the X2 algorithm was then compared to the output from the algorithm on X1. To mitigate any timing issues, the same global clock was used for both

X1 and X2. After minor modifications to the X1 interface code, proper operation was observed with the same output from X1 and X2. A fault injection technique was then developed to verify proper detection of an error.

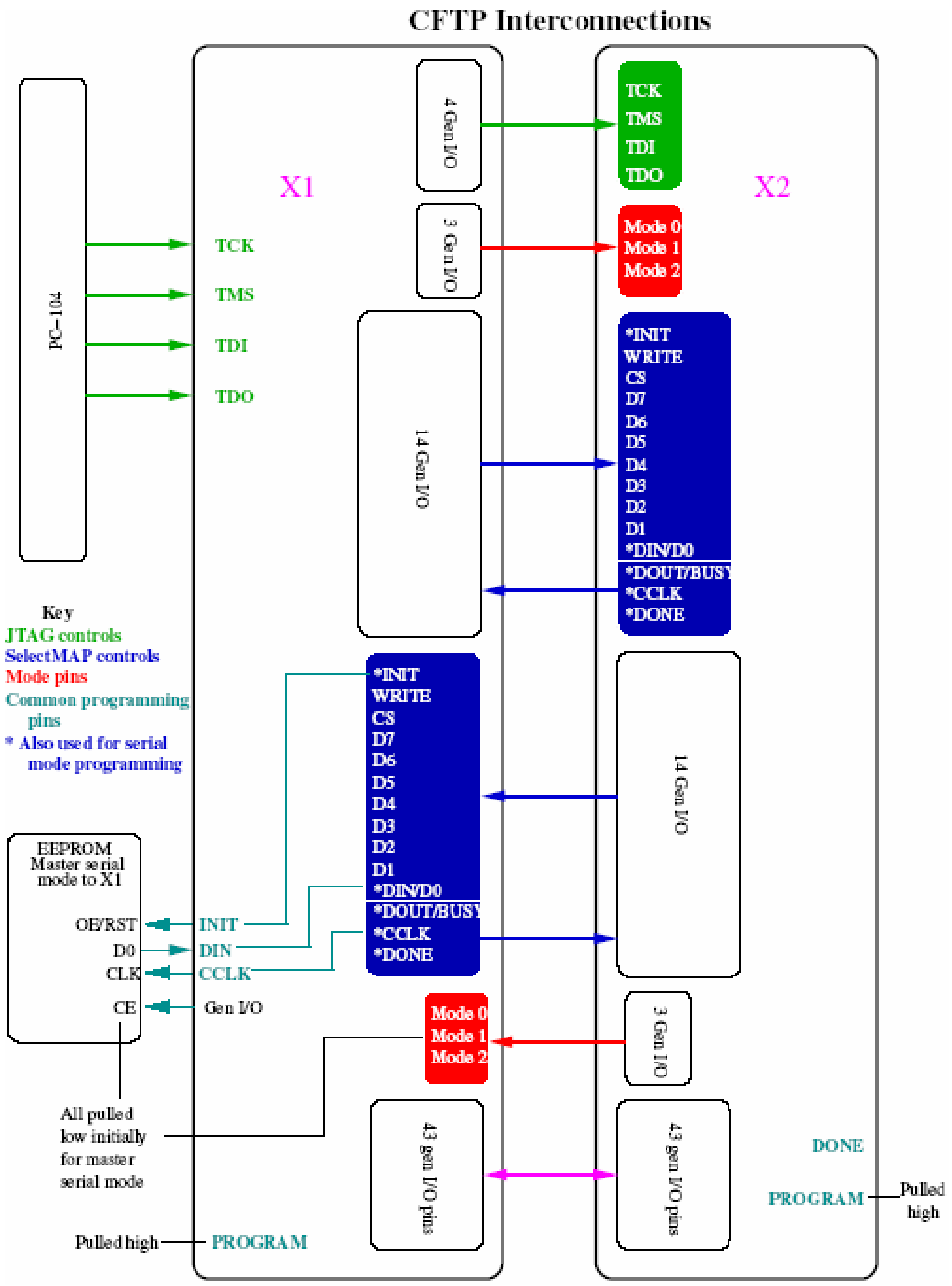


Figure 4. CFTP Interconnections (From [17])

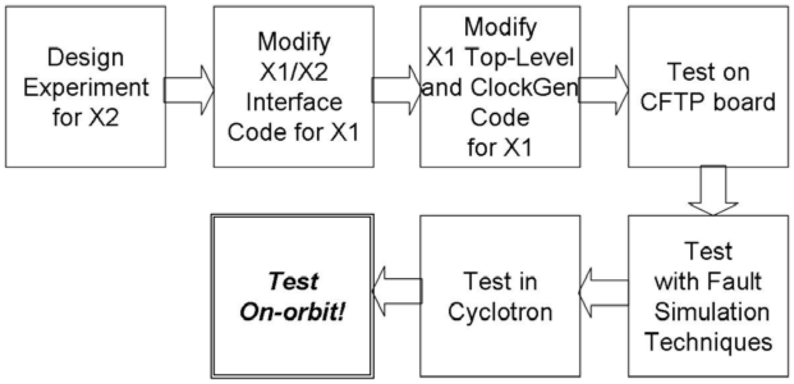

Design Experiment for X2
Modify X1/X2 Interface Code for X1
Modify X1 Top-Level and ClockGen Code for X1
Test on CFTP board
Test with Fault Simulation Techniques
Test in Cyclotron
Test On-orbit!

Figure 5. Experiment Design and Test Process

The first method of fault injection utilized FPGA Editor in the Xilinx ISE Software. The configuration diagram was opened, and a utilized component was changed to produce a predictable error. A partial bit file was then generated with the bitgen command and the original and the new "faulty" designs. This partial bit file was a configuration bit file based on the difference between the two designs. This bit file was programmed onto X2 via JTAG while the experiment was operating. The experiment error counter started to increment, and the output from X2 differed from the X1 output, as expected. Fault injection using FPGA editor was also used for the shift register designed for radiation testing, and the results will be discussed in the next section. This method was useful for determining proper operation of the experiment and to verify the error counting and reporting module was operating as desired. Unfortunately, this method is time-consuming.

A fault injection tool was created by Surratt utilizing the Java-based JBits 2.8 by Xilinx [17]. (The Virtex II FPGA required Jbits 3.0.) JBits is an application programming interface (API) into the Xilinx family FPGA configuration bitstream. JBits is tile based, where different tiles represent CLBs, I/O Blocks (IOB), I/O Interfaces (IOI), Block RAM (BRAM), BRAM Interfaces (BRAMI), Global Clocks (GCLK), and unconfigurable blocks. A tile map of the Virtex I FPGA is seen in Figure 6. The fault injection tool can be used to select a configuration bit to change or can be set up to inject a given number of random configuration bit faults over the entire FPGA, similar to what could be expected during radiation testing. The program creates a full "corrupted" bit file, which is programmed into X2 via the JTAG port. This tool was primarily used to anticipate what faults would be seen during radiation testing. These faults would be read during SelectMAP readbacks and classified as errors when the fault affected program operation. This tool was developed just prior to radiation testing, but yielded interesting results in a short time. One such result was the observation of double reconfigurations, when only one would be expected. Slight modifications were made to the X1/X2 interface code for all experiments and tested again. Double reconfigurations for the CORDIC algorithm were fixed, but the Shift Register program would still get stuck in a reconfiguration loop. It was decided to proceed with the programs as is and observe and compare results during radiation testing. Radiation testing, in turn, produced similar results, which verified the validity of the fault injection tool.

Radiation testing is also performed during space qualification tests to verify a component can withstand the radiation environment of space in terms of total ionizing dose and Single Event Latch-up (SEL) [8]. To complete the integration and testing overview, a brief description of the space qualitication process and environmental testing for NPSAT1 and CFTP integration is provided. Focus is then shifted to radiation testing for CFTP and the observation of radiation effects (SEUs) on the operation of three different programs.

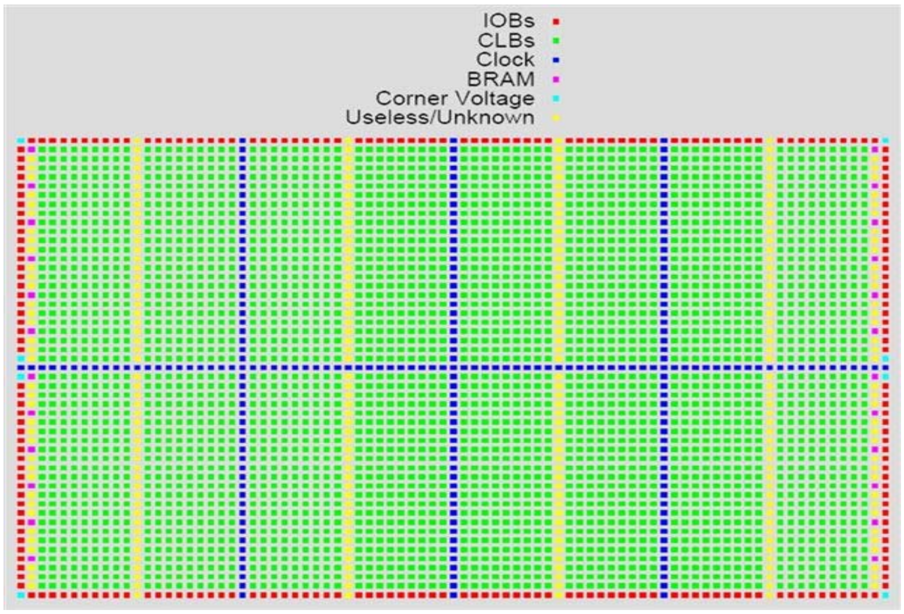


Figure 6. Tile Map for Virtex I XQV600 (from [17])

4. Space Qualification/Environmental Tests

The exponential increase in information-technology performance has led to a change in how systems are qualified for space. The general progression can be summarized with the following categories [23]:

1) component - individual discrete, integrated circuits are qualified
2) manufacturing process - manufacturing line is qualified
3) system design - system is qualified
4) design approach/design tools - hardened by design

Initially, component qualification was dictated by Military Specification Standards (MIL-STD). Now, the focus is on performance specifications, and manufacturers qualify the process, not the component. The following lists what manufacturers must show to demonstrate a given radiation hardness level [23]:

1) Model verification, design rule verification and performance verification
2) Statistical Process Control (SPC), Technology Characterization Vehicle (TCV)
3) Evaluation circuits
4) Parametric Monitors
    a) Particular for GaAs devices under total dose, neutron or proton fluence , test structures must monitor sheet resistance, isolation, Field Effect Transistor (FET) parameters.
    b) Tests shall also examine Dose-rate latchup, dose-rate upset, SEE, Total Ionizing Dose (TID) and displacement damage by protons or neutrons.

The last item highlights the more common purpose for radiation testing. As previously mentioned, radiation testing for CFTP will focus on SEUs and the effect on circuit operation. In addition to radiation testing, components also undergo additional environmental testing. The above processes are primarily applied to electronic components. Further testing is required for circuit cards and system designs. Environmental tests are designed to validate these designs.

Environmental tests are designed to verify the survivability of the component and system for launch and space environments. For example, the array of testing for NPSat-1 is listed below [24].

1) Functional/Verification Test

   Functionality will be verified before and after every environmental test.

2) Static Loads Test

   The Engineering Development Unit (EDU) will undergo a static loads test for qualification.

3) Random Vibration Test

   The EDU will undergo three-axis vibration testing for qualification.

4) Low-Level Sine Sweep Test

   This test determines vibration modes and will be performed before and after each vibration test.

5) Mechanical Shock Test

   A shock test on the EDU will be performed to verify survival from the spacecraft separation system.

6) Thermal-Vacuum (TVAC) Cycling

   Bad solder joints on electronic components will be detected during this test.

7) Electromagnetic Compatibility (EMC) Test

   This test verifies that the spacecraft electronics are not susceptible to damage by radio frequency (RF) emissions during pre-launch, launch and orbital environments.

8) Mass Properties Test

   Center of Mass and Moments of Intertia data will be determined for attitude control.

9) Magnetics Test

This test is performed to calibrate the magnetometer to be used for the attitude control subsystem (ACS).

System testing for CFTP will be performed at the box level prior to integration with NPSat-1. Box testing will consist of vibration testing, thermal-vacuum cycling, and electromagnetic interference (EMI) testing. Vibration testing will first be accomplished in the orientation of launch with just the base plate and a component-less circuit card to measure deflection. Launch orientation of NPSAT1 was shown in Chapter I, and the location of CFTP within NPSAT1 is shown in Figure 7. This data will be analyzed to verify no component damage will occur due to incidental contact with other parts of the box. The vibration test is then repeated with the processor card inserted in place of the blank circuit card. This vibration test will verify the components are securely fastened to the respective circuit cards, and all components are properly fastened to the base plate. This also validates the engineering design of the CFTP project.

After initial vibration testing is satisfactorily completed (no component loss or damage), the power supply is added to the base plate and the processor configuration is placed in the vacuum chamber for TVAC cycling. As mentioned previously, TVAC cycling locates bad solder joints that may cause a component to dislodge during launch or prevent a signal from passing through the pin. After TVAC cycling, the entire CFTP experiment is assembled in the box and put through another vibration test. The final environmental test for CFTP prior to integration with NPSat-1 is the EMI test. EMI is mitigated first with part selection and design. Shielded twisted pair (STP) wiring is used to minimize conducted EMI with other components in the spacecraft. Additionally, external EMI is mitigated by using an isolated power supply, which also minimizes EM leakage. Radiated EMI is mitigated by containing the CFTP experiment in an aluminum box.

CFTP is ready for integration with NPSAT-1 upon completing this box-level testing. After launch in October 2006, CFTP is ready to run experiments on-orbit and provide data to correlate with ground radiation testing.

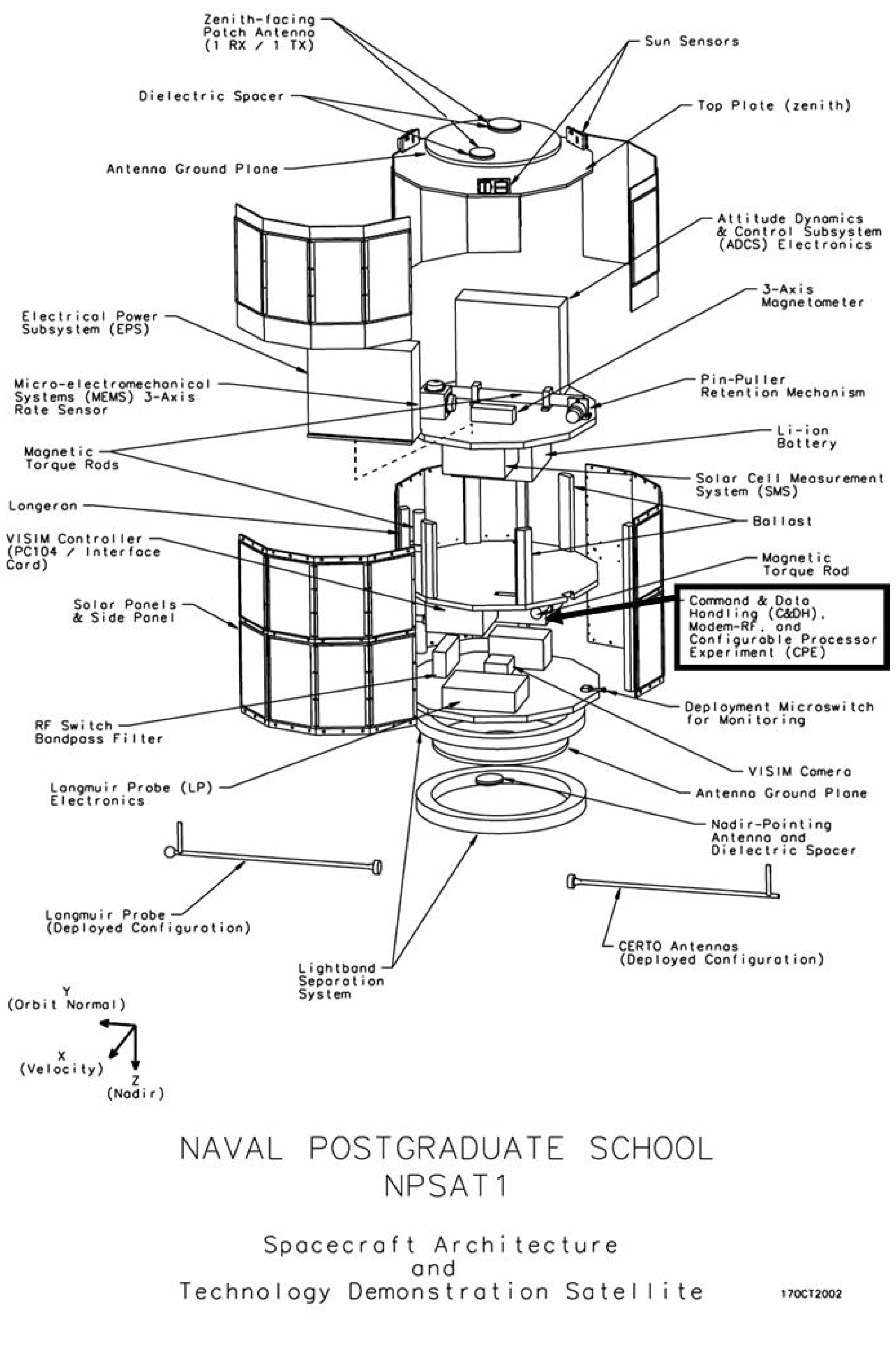

Zenith-facing Patch Antenna (1 RX / 1 TX)
Sun Sensors
Dielectric Spacer
Top Plate (zenith)
Antenna Ground Plane
Attitude Dynamics & Control Subsystem (ADCS) Electronics
3-Axis Magnetometer
Electrical Power Subsystem (EPS)
Micro-electromechanical Systems (MEMS) 3-Axis Rate Sensor
Pin-Puller Retention Mechanism
Magnetic Torque Rods
Li-ion Battery
Solar Cell Measurement System (SMS)
Longeron
Ballast
VISIM Controller (PC104 / Interface Card)
Magnetic Torque Rod
Solar Panels & Side Panel
Command & Data Handling (C&DH), Modem-RF, and Configurable Processor Experiment (CPE)
Deployment Microswitch for Monitoring
RF Switch Bandpass Filter
VISIM Camera
Antenna Ground Plane
Langmuir Probe (LP) Electronics
Nadir-Pointing Antenna and Dielectric Spacer
Langmuir Probe (Deployed Configuration)
CERTO Antennas (Deployed Configuration)
Lightband Separation System
Y (Orbit Normal)
X (Velocity)
Z (Nadir)
NAVAL POSTGRADUATE SCHOOL
NPSAT1
Spacecraft Architecture and Technology Demonstration Satellite
17OCT2002

Figure 7. NPSat-1 Launch Orientation (From [25])

5. Radiation Testing

Radiation testing was conducted in separate stages. The first stage was completed when FPGA Editor was used to inject a fault to verify proper operation of the given experiment. The JBits-based fault injection tool was then used to generate multiple faults during one run period. These results provided an excellent knowledge base for observing real-time radiation test results at UC-Davis, which were conducted in accordance with Appendix A. The final stage of radiation testing then occurs after October 2006, with NPSAT-1 and MidSTAR-1 on-orbit. Fault injection techniques were slightly different between the two development boards.

a. CFTP Virtex I vs. CFTP Virtex II

The design detailed in [8] will fly on NPSat-1 and MidSTAR-1. A second board was also built by the Naval Research Laboratory and Silver Engineering for experiment development and radiation testing. This second board, referred to as CFTP-2, utilizes a Virtex II, 6-million-gate-equivalent FPGA (XC2V6000) for X2. From the interface design standpoint, one significant difference between the two boards is the use of different variants of the same flash memory chips (Intel Advanced Boot Block vs. Intel Advanced Boot Block+ [blocks are auto-locking]). The flash memory on CFTP-1 is bottom boot; the first eight blocks are eight Kbytes each. The flash memory on CFTP-2 is conversely, top boot. This difference required a modification to the computer code for the flash memory. Additionally, a second flash chip was added due to the increased size of a configuration file for a Virtex II FPGA. These modifications were successfully tested on CFTP-2. Another difference between the two boards is the lack of SDRAM memory on CFTP-2. With the increased capacity of the Virtex II FPGA, the SDRAM was deemed unnecessary. The Virtex II also has a different pinout and array address naming convention, which yielded a different user constraint file. A user constraint file is used to specify physical constraints that the user wants to impose, such as a specific signal tied to a specific pin or a specific program to a specific area of the FPGA. The constraint files are located in Appendix D. The major difference, though, is the order of magnitude size difference between the two FPGAs. This enabled the experimenters to run the PIX program on the CFTP Virtex II board. Once these differences were understood, the JBits-based fault injection tool was developed for both CFTP boards.

b. Ground Testing

The purpose of radiation testing for CFTP is to determine how an algorithm/experiment is going to operate when SEUs occur. The fault simulators can simulate these SEUs by "flipping" a bit from a "1" to "0", or vice versa, and experimenters can observe the operation/performance. As previously mentioned, using FPGA Editor to change an item in the configuration drawing was a tedious and time consuming process. The configuration file (.ncd) is an output from the place and route function in Xilinx ISE. This file was opened in edit mode, and some function of a LUT, FF, MUX, routing, etc. was changed. For example, for the shift register the LUT block was changed from a Xilinx macro-function utilizing the RAM memory elements to a LUT outputting a constant "1." The LUT blocks are highlighted squares on the left in Figure 8. A bit difference file was then created and used to program X2 via JTAG to simulate the SEU.

As an alternative, the JBits-based fault simulator was designed with a graphical interface, displaying a tilemap as shown in Figure 6. Specific bits can be changed, and the graphical interface makes it easy to observe fault occurrence. The first method of injecting a fault was done by clicking on a particular tile. The program would then randomly flip a bit within that tile, program X2 with the new "faulty" configuration, and display the actual location of the injected fault as a red circle. After a fault triggered a reconfiguration, all corrected faults were displayed as triangles. A picture of the tile map with uncorrected (circles) and corrected (triangles) faults is shown in Figure 9. A script was also designed to generate faults at a specified rate to observe multiple faults over time [17].

The third stage of ground radiation testing was to test the developed algorithms in the cyclotron at UC-Davis. The test plan used for this testing is in Appendix A. The cyclotron at Crocker Nuclear Laboratory bombarded CFTP with 63 MeV protons to produce SEUs. The performance of each algorithm was then monitored for fault and error handling. One minor note for radiation testing is that the Virtex I and Virtex II FPGAs used for X2 are not radiation hardened. The manufacturing difference between hardened and unhardened Xilinx FPGAs is the addition of an epitaxial layer for the radiation variant. The epitaxial layer mitigates the likelihood of SEL, as well as increasing the total dose tolerance of the FPGA [26]. This should not affect the occurrence of SEUs, but may be an area for future studies. Results from radiation testing were then compared to the results from the fault injection tool as described earlier. Radiation results will also be used to establish a baseline for expected SEUs and experiment operation on-orbit.

c. On-orbit testing

All testing described to this point is intended to validate designs and verify that systems will withstand the space environment. Although these tests are intended to be as similar to space conditions as possible, spacecraft engineers are hesitant to utilize newly developed components or technologies without demonstrated performance in space. A common method for demonstrating this performance is through an experimental test bed, such as that provided for CFTP in NPSat-1 and MidSTAR-1. The CFTP design of using a control and experiment FPGA is not new, but has only been demonstrated in space on the Adaptive Instrument Module (AIM) launched on the Australian FEDSAT-1 on 14 December, 2002 [27]. In addition to demonstrating the use of a control and experiment FPGA, CFTP is designed as a test bed to run different algorithms to test different reliability strategies and reconfigurability for FPGAs. Operational data of different algorithms has been collected through ground testing. The same algorithms will be run on-orbit to validate ground testing results. More importantly, a TMR processor (PIX must be made smaller to fit on the flight-version CFTP Virtex I) and other algorithms for configuration-error detection and scrubbing techniques can be run to determine best practices and designs.

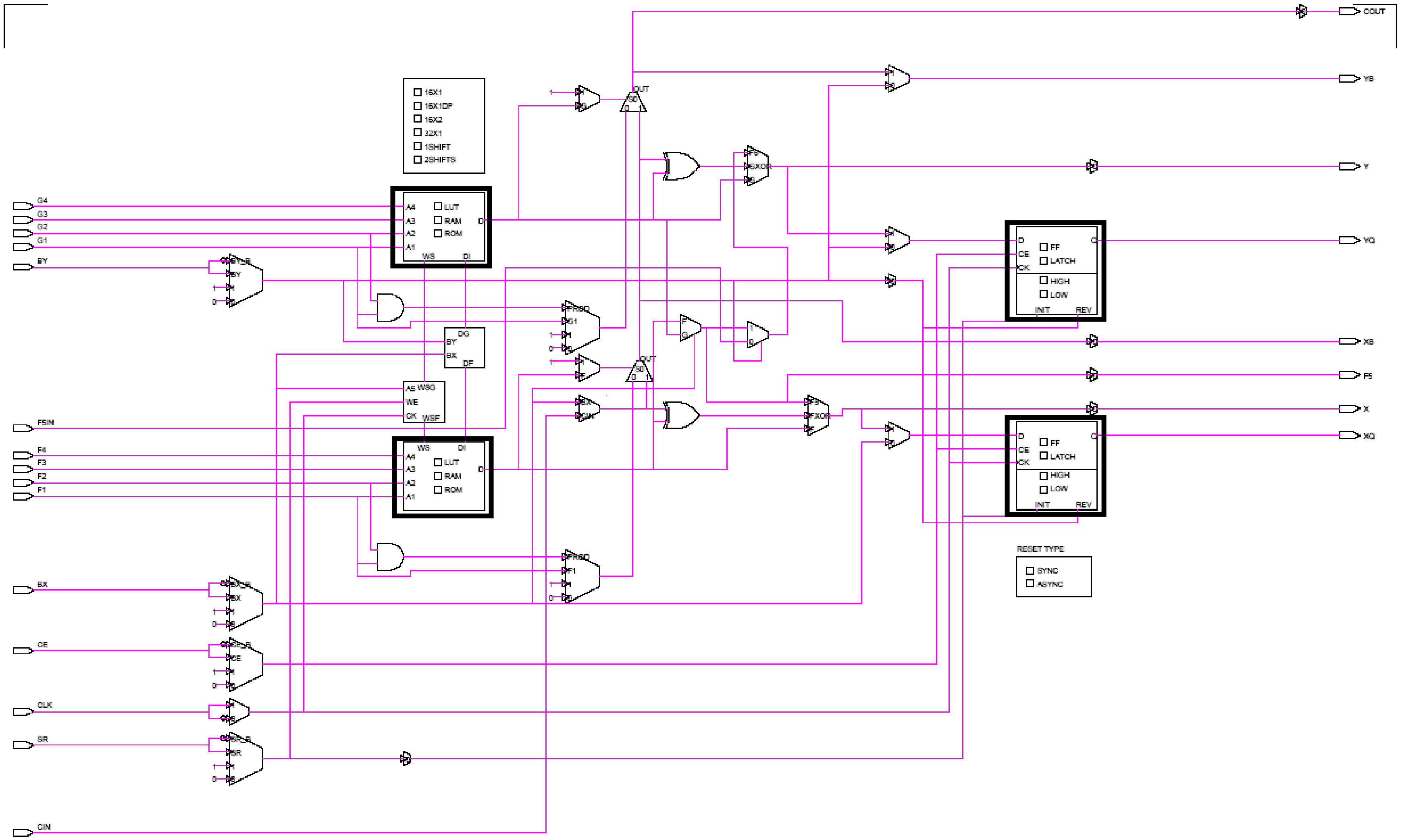


Figure 8. Xilinx Virtex I Slice (From [14])

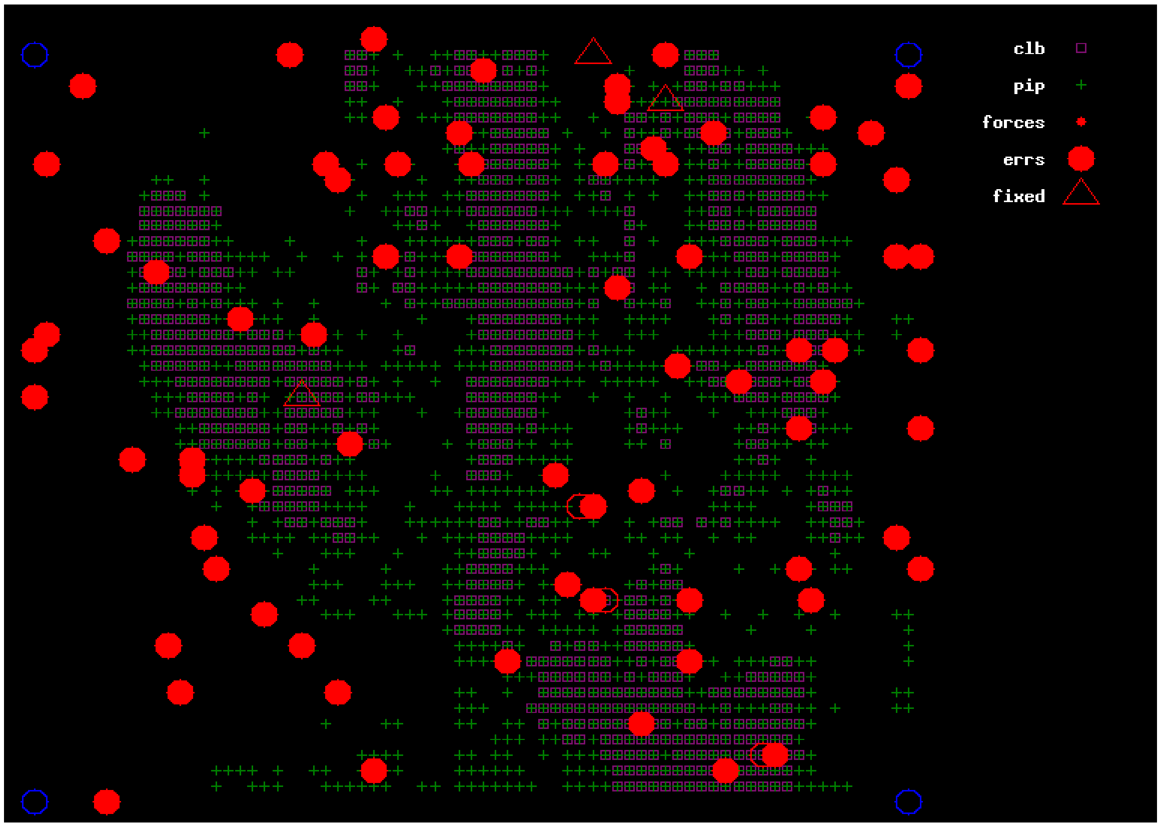


Figure 9. Tile Map Display - JBits and Radiation Testing

The inaccessibility of space is precisely why CFTP-like systems are an ideal solution for computer processing in space. When configuration errors are detected, a new configuration (or partial configuration) can be programmed with little to no loss in satellite operations. In this vane, radiation testing for CFTP will help determine best-practices to detect configuration errors, correct them, and ensure proper operation of the processor (and other components of the satellite). Preparations for radiation testing are outlined in the next chapter.

IV. RADIATION TEST PREPARATION

A. CFTP INTERFACE PROGRAM

The interface software for CFTP was developed in a modular fashion to enable simple integration with any X2 experiment. One key function within the X1 code was to generate error reports and print the output from SelectMAP readbacks. During development, “heartbeat” error reports were set to print every three seconds. This would enable the experimenter to verify that his program was still operating and had not stuck in a steady state, such as could be caused by an SEL. SelectMAP readbacks were programmed to do a comparison check with the configuration in the Flash memory every 30 seconds. This would allow the experimenter to verify the desired number of SEUs was occurring. With a reporting program in place, the next step was to develop a program/circuit to help set the proper proton flux level for the desired SEU rate and validate results from previous radiation testing with Xilinx FPGAs. A shift register experiment was then developed for X2 to observe SEUs and ensure the proton flux was at an appropriate level for data analysis.

B. SHIFT REGISTER

A shift register is a simple circuit that shifts 1s and 0s through a series of flip-flops. A shift register, which provided for intermediate data comparison, was developed for initial testing at the UC-Davis cyclotron.

1. Purpose

The desired SEU rate for CFTP radiation testing was arbitrarily chosen to be 2-5 SEUs per minute. This rate would be high enough to observe that SEUs were occurring, but low enough to enable the experimenters to determine if their algorithm was operating properly and decipher the test results. A simple algorithm was desired to cover the maximum amount of physical area on the FPGA chip to allow for the best probability of a configuration fault manifesting itself as a data error. Maximum area is defined here as utilizing the maximum number of slices on the FPGA. Each CLB consists of two slices, pictured in Figure 8. A shift register was chosen due the simplicity of the shift register and the ability to obtain limited SEU location data while the circuit is operating. The bit length of the shift register was then increased or decreased to maximize the area coverage of the FPGA.

2. X2 Design

The implementation of the shift register needed to allow for SEU detection, provide limited location information, and provide for simple length adjustment. Xilinx FPGAs can use the Look Up Tables (LUTs) as a 16-bit memory element for shift register implementation with the SRL16 macro. The first shift register design used only these macros. The second shift register design incorporated flip-flops interleaved with the SRL16 macros to increase the area usage of the FPGA. The final design disallowed use of the SRL16 macro, forcing the ISE program to utilize only flip-flops.

The input for the shift register comes from a maximal-length 15-stage LFSR instantiated on X1. This generates pseudo-random 0s and 1s continuously, until the program is stopped. Because X1 is not in the proton beam, the output from the LFSR was assumed to be "safe" from SEU. Possible SEUs could occur between the input at X2 and the input of each shift register, and this possibility is addressed in the detection design.

a. Detection

Detection of SEUs was implemented using a comparison-based technique using XOR gates between two different register "trains" [28]. The first design used the SRL16E (E is for clock enable) for the shift register. To most efficiently compare two different register trains (further referred to as the a-side and b-side), the sixteenth bit (the output of each SRL16E) of each train was routed to an XOR gate. If any comparison output produced a "1", indicating that an SEU had occurred in one of the trains, a signal "sumdiff" was pulled high, as illustrated in Figure 10. This shift register module was then copied 15 additional times to create 16 different columns.

It is also possible that an SEU could occur between the input of the LFSR bit stream to X2 and the start of the shift register trains, previously mentioned, or conversely, between the output of the shift registers and the output of X2. This would lead to a fault propagating simultaneously through the a-side and b-side of the shift register train, undetectable to the XOR circuitry. A copy of the shift register was initially implemented on X1 to allow for comparison of the sixteen train outputs from X2 to detect such an SEU. This implementation created some anomalous results during program operation, so the function was moved to X2. The output from the a-side of the trains were compared against each other using a 16-input XOR. If the output was a “1”, a separate error counter would increment. It is recognized that two SEUs could propagate to produce a “0”, or an undetected fault, but CFTP is designed with the assumption that two or more SEUs will not occur at the same time. The implementation of this secondary error detection function is illustrated in Figure 11.

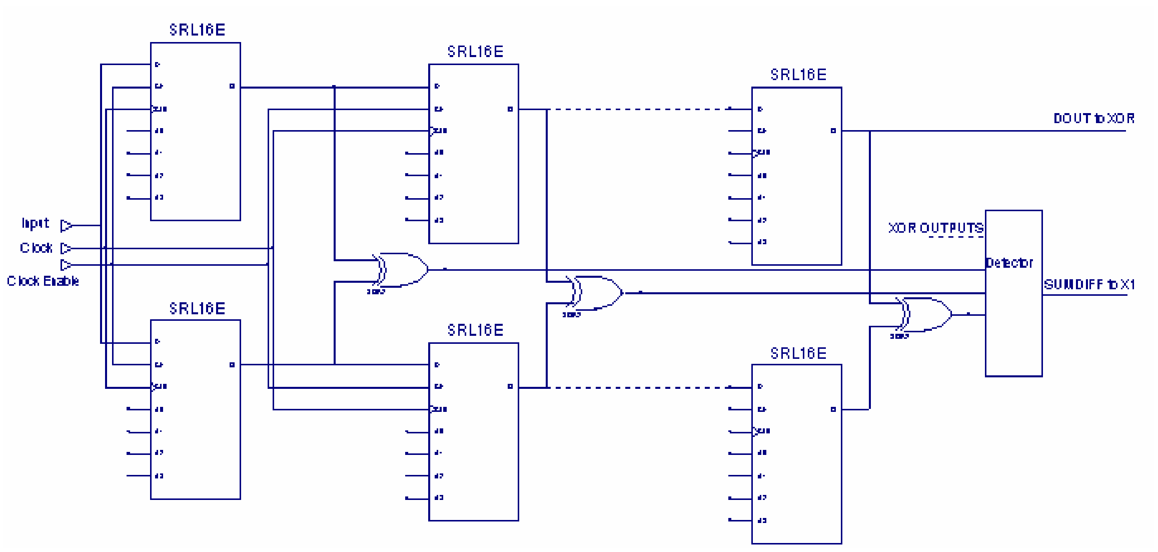


Figure 10. Shift Register Implementation

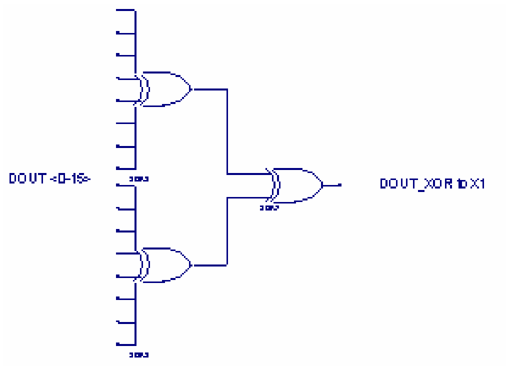


Figure 11. Secondary Error Detection

b. Location/Constraints

To address the ability to discern some location data for SEUs, the output was designed to reflect a given area of the FPGA. Sixteen areas were chosen as there are 42 auxiliary input/output (I/O) pins between X1 and X2 for programmer usage. Because the shift register initially used 2 outputs for each copy of the shift register, sixteen would allow for the maximum number of copies of the shift register. At the same time, sixteen columns were a small enough number to easily interpret the data reports during the shift register operation.

Three patterns were considered for generating additional copies of the shift register. The first was a 4x4 checkerboard pattern, where the report output would be labeled in a grid pattern A1, A2,…,A4, B1, …, D4. The other two options were sixteen columns or sixteen rows, with all three options pictured in Figure 12. The checkerboard pattern had a potential to yield confusing results, based on interpreting the error reports, compared to the simpler column or row format. A majority of the auxiliary I/O pins for X2 were located at the top. So in an attempt to increase place-and-route efficiency by the ISE software, and possibly increasing the length of the shift registers, the column pattern was chosen. This also provides for an easy report layout.

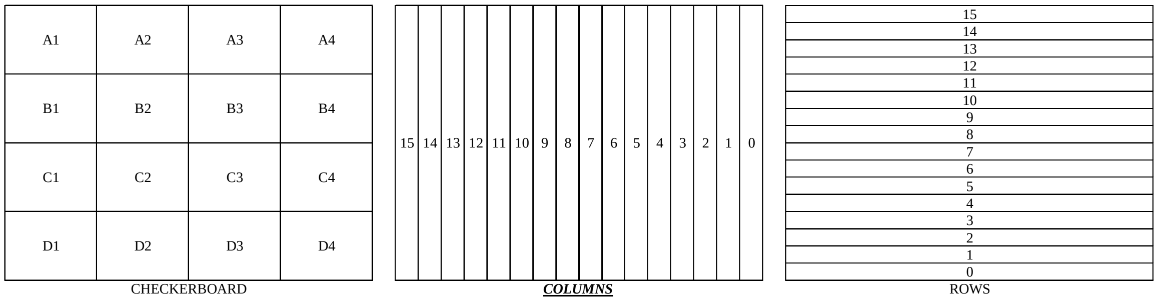


Figure 12. Shift Register Generation Patterns

The VHDL "generate" command was used to make sixteen shift-register trains. The shift-register module and X2 top-level code are listed in Appendix D. These shift-register banks were further defined in a constraint file in the Xilinx ISE software to occupy the columns as defined in Figure 12. The constraint files for the Virtex I and Virtex II boards are both listed in Appendix D.

c. SRL16 Implementation

The length of the shift-register train was defined with a global variable. This made it easy to increase the size of the shift registers to maximize area coverage of the FPGA with different implementations for the Virtex I FPGA or (by enlarging the size) for the larger Virtex II chip. As previously mentioned, the initial shift-register design used the SRL16 macro. More specifically, the SRL16E, which has a clock enable function, was used. This clock enable function was used to stop the shift register when performing a SelectMAP readback. The SRL16E macro uses the four address lines of the LUT as configuration control to utilize the SRAM cells of the LUT as a 16-stage delay, where each stage can be tapped. The data shifts on each clock pulse. [29].

Because the SRL16E performs as a 16-bit shift register, the output of each SRL16E was compared for SEU detection as described earlier. Additionally, 16-bit increments were used when changing the size of the shift register to maximize FPGA coverage. The tested version of the shift register using only SRL16E macros (no flip-flops) contained 2400 bit shift registers and used 88 percent of the slices. Area constraints during the synthesis process (netlist creation) were a limiting factor that prevented using 100 percent of the slices. Length and slice usage for the other two versions of the shift register are summarized in the next section.

3. Shift Register Variations

Two additional variations of the shift register were developed to analyze the data sensitivity to the LUTs (SRL Macro) and flip-flops. The intent behind the second shift register design was to force the Xilinx software to utilize the LUTs as SRL16E macros (like the original shift register design) and the flip-flops, which were not utilized in the original design [30]. To do this, the sixteenth AND seventeenth bits were XOR'd and routed to the detector. The design, labeled SRL+1 for SRL plus one bit, is shown in Figure 13. The third variant utilized only flip-flops by disabling the SRL16 macro function. Different designs and the two different Virtex chips led to different length shift registers. Table 1 contains a summary of the different shift register sizes for each version, along with the slice usage. The percentage of slice usage varies varies from 74 to 97 percent between the various designs on both Virtex chips. The differences are believed to be due to the implementation of the SRL16 macro versus flip-flops on the same Virtex model, and different structure of a slice on the newer Virtex II.

Due to the modular design of the shift register experiment, only the shift register module needed to be changed. All top-level code was unchanged, and the VHDL code for the alternate shift register designs is listed in Appendix D.

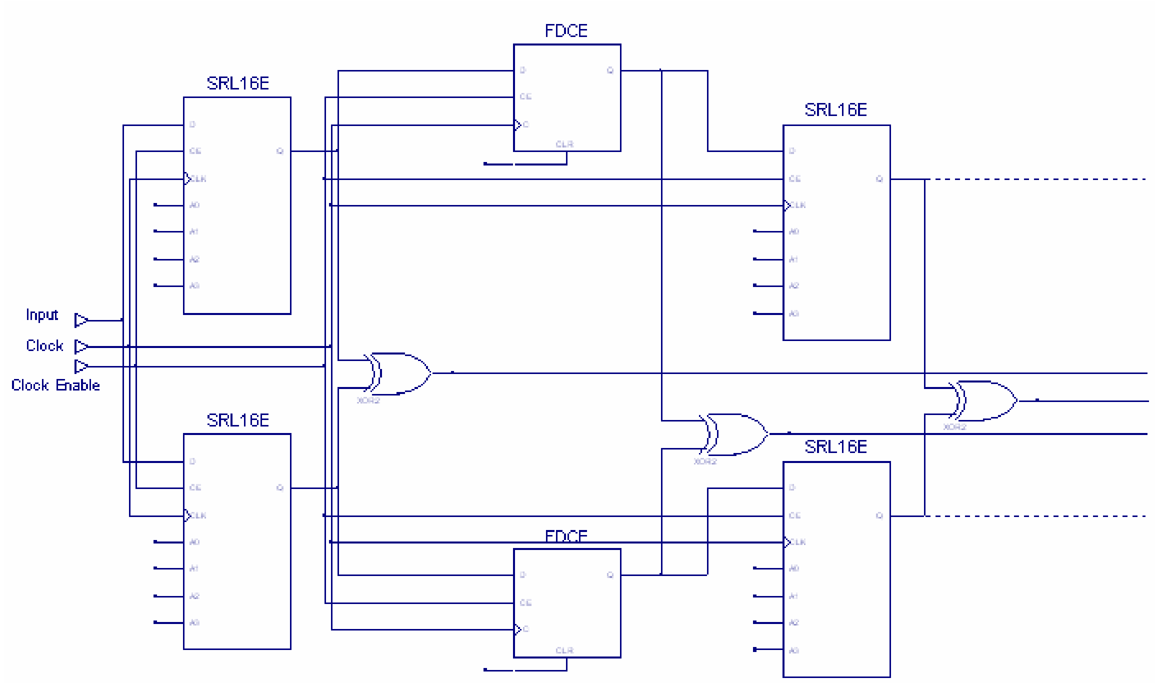


Figure 13. SRL16E and FF Shift Register Design (SRL+1)

Table 1. Shift Register Length (bits) and Slice Usage (%)

	Virtex I		Virtex II	
Shift Register Design	Length	Slice Usage	Length	Slice Usage
SRL16E only	2400	88	12000	91
Flip-flops only (no SRL)	320	79	1500	74
SRL16E and flip-flops	2040	89	8500	97

Because proper operation and fault detection reporting was observed with the first shift register design, fault injection using FPGA Editor was bypassed for the additional variants of the shift register. All versions were tested with the JBits-based fault-injection tool, and the results are discussed in Chapter V along with the results from cyclotron testing.

4. X1 Design

The interface VHDL code for X1 was created by Surratt [17]. Shell files for the X1 top level code, X1-X2 interface code, clocking code, and the constraint file were modified to operate with the experiment program. SelectMAP and PC/104 modules are standard for any experiment and are listed in Appendix D.

The first task was to create the X1-X2 interface code (listed in Appendix D). For the shift register, this included creating the LFSR to generate the bitstream, a parallel shift register, and a counting/comparator module for reports. The LFSR was based on Gulotta's example and modified for the following polynomial: $X^{15} + X + 1$ [29]. This polynomial minimized the tap points to two, but was large enough to generate a pseudo-random sample of bits. The next task was implementing a copy of the shift register module on X1. When initial simulations showed anomalous events regarding the comparison between the X2 shift registers and the X1 copy, this function was moved onto X2.

The next addition to the interface code was the counting module. This consisted of seventeen variables: one to count whenever a shift register train did not agree with the rest and one for the XOR detector output of each of the sixteen columns. "Heartbeat" reports were printed out to PC/104 every 3 seconds, and a SelectMAP readback was completed every 30 seconds. If any counter reached 128 errors (hex"80"), the part was forced to reconfigure. Sample output reporting from the shift register is shown in Table 2. The signal dout allows the experimenter to verify the LFSR is sending 1s (01 01) and 0s (00 00), and sumdiff is a hexadecimal representation of the XOR detector output from each column. For example, a detected data error in the third column from the right would read hex"00 04". The signal dout errcnt is the XOR output from the dout output of each column, which detects that a possible data error that propagated simultaneously through a shift register train. Lastly, the counters for each column are displayed relative to the FPGA, meaning an incrementing counter in one of the right columns indicates an SEU on the right side of the FPGA.

Table 2. Sample Shift Register Report

```
Selectmap Readback:
Read: 40, Expected: 00, Mask: 00, Location: 027890, MJA: 28, CLB column, left half
Read: e6, Expected: e4, Mask: 00, Location: 059b73, MJA: 64, CLB column, left half



21:57:20
dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:22
dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:25
dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:28
dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:31
dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:34
dout: 01 01    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:37
dout: 01 01    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:40
dout: 00 00    sumdiff: 00 04    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 81 00 00
21:57:43

Selectmap Readback:
Read: ba, Expected: fa, Mask: 02, Location: 0076f6, MJA: 6, CLB column, left half
Read: 40, Expected: 00, Mask: 00, Location: 027890, MJA: 28, CLB column, left half
Read: 3d, Expected: 3f, Mask: 00, Location: 043eae, MJA: 49, CLB column, right half
Read: e6, Expected: e4, Mask: 00, Location: 059b73, MJA: 64, CLB column, left half



21:57:44
Selectmap Reconfig...

dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00
21:57:46

Selectmap Readback:
```

```
21:57:50
Selectmap Reconfig...

dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:53
dout: 01 01    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:56
dout: 01 01    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:57:59
dout: 00 00    sumdiff: 00 00    dout errcnt: 00    Col(15:0) errcnt: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:58:02
```

After the interface code was completed, the clocking code (clockGen.vhd) was modified. The 25 MHz clock was initially used to the test the shift register program. Analysis of the timing revealed data paths that were taking up to 79 nanoseconds (ns), more than the 40 ns duration of the 25 MHz clock. The clock was then divided down once to produce a 12.5 MHz clock. The copy of the shift register train on X1 was removed after dividing the clock, which produced a maximum signal delay of 23 ns. Although the timing constraint of the 25 MHz would then have been met, it was decided to keep the clock at 12.5 MHz. The clock code is shown in Appendix D.

The last item needed for the experiment to operate was the X1 top level code (cftp_x1.vhd), which provided overarching control for all the modules on X1. The primary modifications to this code were to ensure the proper signals to and from X2 were declared for the X2 experiment. This code is also shown in Appendix D along with the constraint files for the Virtex I and Virtex II boards.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RADIATION TEST RESULTS

A. FAULT INJECTION

1. FPGA Editor

FPGA Editor allows the user to manually change the configuration of an FPGA. This tool was used with the SRL16-only version of the shift register. The final output of the b-side register is XOR'd with the a-side output to produce the sumdiff bit that is sent to X1. The a-side output is also compare with the a-side output from the other 15 copies of the shift register. To produce a singular error, the final output of the b-side register was changed to a constant '1'. This was accomplished by changing the function of the LUT from a SRL16 shift register to a LUT that outputs a '1'. This only produces an error whenever the output bit from the a-side is a '0'. The sumdiff output reads x"80 00," indicating the far left column has a data miscompare. The dout errcnt remains x"00 00," because the a-side output is still producing the same output as the other 15 shift register trains. The other observable, Col(15:0) errcnt, increments.

This new configuration was saved, and a bit difference file was created using bitgen [14]. This bit file was then programmed onto X2 via JTAG. The expected results described in the previous paragraph were observed. This process deomonstrated that the error counting function in the X1 programming was operating properly. Because the other two versions of the shift register use the same X1 code and X2 top-level code (only the shift register module was changed!), they were not tested in this manner.

2. JBits

JBits was programmed to inject random errors in the Virtex I design to establish a baseline for fault occurrence and reconfiguration prior to radiation testing. (The fault injection for the Virtex II using JBits 3.0 was still under development.). All three shift register versions were tested. Using a modified MATLAB program designed by Josh Snodgrass, the results were first analyzed for 1-to-0 and 0-to-1 transistions [31]. The MATLAB script is located in Appendix D. A preponderance of 0-to1 transitions were observed for the SRL16-only and flip-flop-only shift registers. (The results from these two versions were sufficient for pre-radiation observations, so the SRL16-plus-flip-flop version was only briefly tested.) The second observation was the number of faults accumulated before reconfigurations occurred. These results are shown in Table 3.

Table 3. JBits Fault Injection Results

Shift Register	0-to-1 (#)	1-to-0 (#)	SEUs (#)	Recon's (#)	Average SEU/Recon
FF-only (run 1)	68	27	95	24	3.96
FF-only (run 2)	55	42	97	14	6.93
SRL16-only	22	4	25	110	0.227
SRL16 + FF	3	2	5	5	1

B. CYCLOTRON

As previously stated, radiation testing was performed on the CFTP-1 using the three shift register configurations and two CORDIC algorithms. CFTP-2 was tested with the same programs and the PIX processor. Results from the CORDIC test runs are incorporated in an overall analysis for on-orbit SEU prediction. Otherwise, only shift register results were analyzed.

1. CFTP-1

Radiation testing was first attempted from 30 August - 1 September 2005. Problems were encountered during initial CFTP set-up. One main problem was that the CFTP board and the TMZ104 processor DC-power-supplies were set with a 1-Amp limit. This caused the voltage to drop when attempting to program X1. The current-limit for the 6-Volt source (set at 5 Volts) supplying power to the TMZ104 processor was increased to 2 Amps. The processor and board then initialized properly, but the voltage for the CFTP board would occasionally drop to 2 Volts, drawing 1 Amp. Because the 25-Volt source (also set at 5 Volts) was limited to a 1-Amp maximum, the current-limit could not be increased. The short-term fix was to restart CFTP and re-run the CORDIC experiment, which was the first working experiment. Later refinement of the interface code fixed the current-voltage problem, along with using separate power supplies with current-limits set at 3 Amps for the processor and the CFTP board.

Various problems were encountered when attempting to run other experiments, too. The shift register was not properly implemented with the interface code, so the

CORDIC algorithm was the only successful experiment run on CFTP-1. The interface code was not yet completed for CFTP-2, so no experiments were radiation tested on CFTP-2. The most significant outcome of the CORDIC testing was the observation of SEUs. The initial setting of the cyclotron produced a flux of 8.48 x 10^6 protons/cm^2-sec, which yielded too numerous SEUs to understand what was happening. So, the setting was decreased to produce a flux of 4.27 x 10^6 protons/cm^2-sec. Proper observation of what was happening with the CORDIC algorithm and radiation-induced faults was still unachievable, so the flux was reduced two additional times. The final setting produced a flux of 8.42 x 10^5 protons/cm^2-sec. The output from the CORDIC algorithm still needed further refinement to better understand what was happening when SEUs occurred, but the flux appeared to be the proper level to observe SEUs approximately every 30 seconds. No further testing was accomplished during this visit.

The August 2005 trip to Davis yielded some lessons-learned. The primary lesson-learned was that the testing team needed to arrive with fully functional bit-files ready to run on the CFTP boards. This was not accomplished for the August testing session. Fully operational bit-files were prepared and tested before the follow-on trip to Davis in November 2005, and non-working experiments were not an issue. One positive lesson-learned was that there were a proper number of people to operate and observe the experiments: one person programming the experiment, one person observing the graphical display, and one person logging/monitoring the current to the CFTP-board. To allow for better understanding while an experiment was operating in the proton beam during future testing, the interface code was modified to output the Major address (MJA) and the right- or left-half of the FPGA of a fault, along with time stamps for the heartbeat reports, readbacks and reconfigurations (see Table 2).

Armed with an array of experiments and an improved test agenda (see Appendix A), testing was completed for CFTP-1 on the first day back at Crocker Nuclear Laboratory, November 14, 2005. The final cyclotron setting from August was used to produce a flux of 8.78 x 10^5 protons/cm^2-sec, which yielded one SEU every 10-30 seconds. The SRL16-only shift register was tested first, and configuration faults appeared every 20 seconds, on average. The cyclotron settings were determined to produce the desired flux, and testing continued. Shift register and CORDIC results were averaged to calculate an SEU rate, defined as the frequency of configuration faults. SEUs occurred every 16.6 seconds on average, for the CFTP-1. SEU rate for each program is shown in Figure 14.

The MATLAB script used on the fault injection results was used for post-testing analysis of the the radiation testing results of the shift-registers on CFTP-1. The summary is shown in Table 4

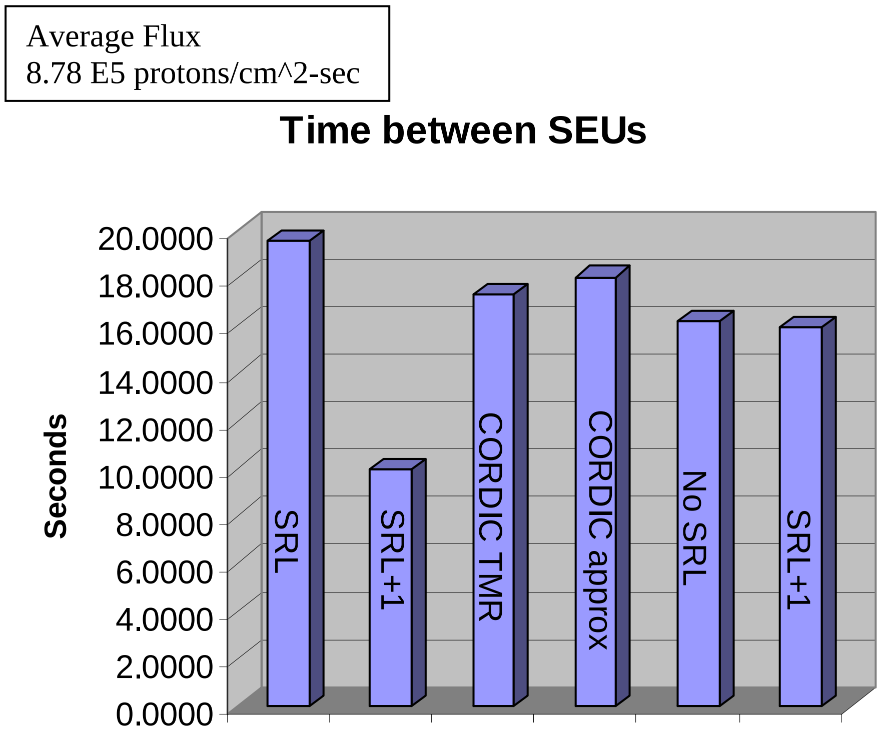


Figure 14. SEU Rates for CFTP-1

Table 4. Radiation Testing Results, CFTP-1

Shift Register	0-to-1 (#)	1-to-0 (#)	SEUs (#)	Recon's (#)	Average SEU/Recon
SRL16-only	16	13	27	7	3.86
SRL16+FF-run 1	15	25	35	7	5
FF-only	13	13	26	4	6.5
SRL16+FF-run 2	76	50	117	19	6.16

No significant pattern was observed in the bit transitions, and there was no direct correlation with the fault injection tool. This could signify the true randomness of radiation testing. All shift registers yielded an error-rate within 28% of the mean; relatively similar considering the different component usage within each design. The error-rate is defined as the frequency that a configuration fault (SEU) triggers a reconfiguration. Note that a data bit flip can produce multiple increments of the error counter as the bit progresses through the shift register, reference Figure 10. This is of note in that a configuration fault that changes the output is assumed to continually increment an error counter, producing a reconfiguration. It is possible that a data bit flipped early in the shift register could produce a reconfiguration, too. A chart of the error-rate is shown in Figure 15. Note that any number of SEUS from 2 to 23 occurred before a reconfiguration was triggered. The reconfigurations with zero SEUs are considered flaws in the fault detection program on X1.

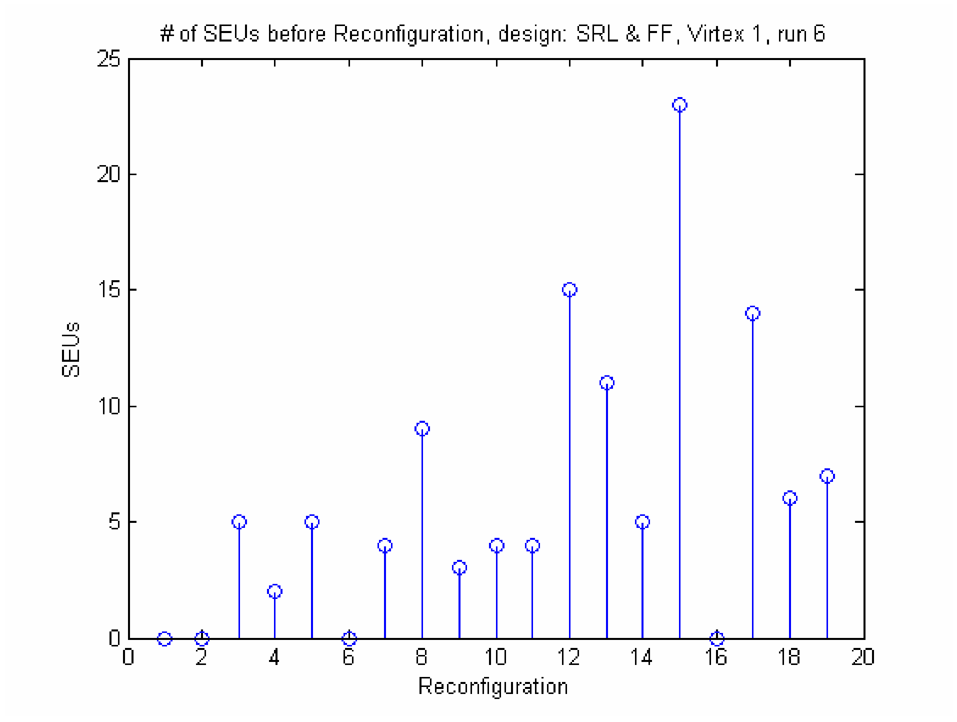


Figure 15. Error-Rate for SRL16+FF (Run Time = 30 minutes)

2. CFTP-2

The second day was dedicated to testing with CFTP-2. The first observation was that faults occurred much more frequently. This was expected with the smaller feature size of the Virtex II FPGA, while using the same proton flux as that used for CFTP-1 testing. SEUs occurred every 1.69 seconds on average with a graph of the SEU rates for CFTP-2 shown in Figure 16. Bit transitions and error-rates are listed in Table 5. There is a preponderance of 0-to-1 transitions, but the cause is unknown at this time. There was also a larger discrepancy in error-rates between the different versions of the shift register. As with CFTP-1, the number of SEUs that triggered a reconfiguration varied as shown in Figure 17.

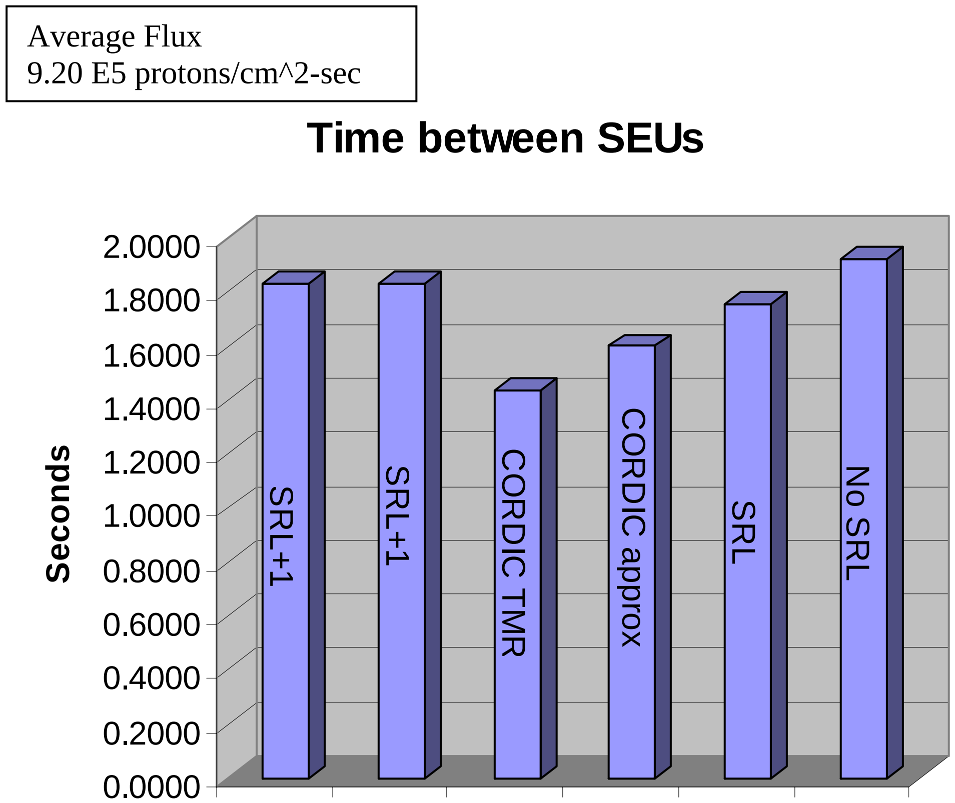


Figure 16. SEU Rates for CFTP-2

Table 5. Radiation Testing Results, CFTP-2

Shift Register	0-to-1 (#)	1-to-0 (#)	SEUs (#)	Recon's (#)	Average SEU/Recon
SRL16+FF-run 1	45	3	47	12	3.92
SRL16+FF-run 2	366	53	414	53	7.81
SRL16-only	164	8	172	19	9.05
FF-only	160	17	172	58	2.96

Radiation-testing yielded varying results for bit-transitions and error-rates, but the SEU rates were similar between different programs run on the same CFTP-board. This information can be used to make a prediction for SEU frequency on-orbit.

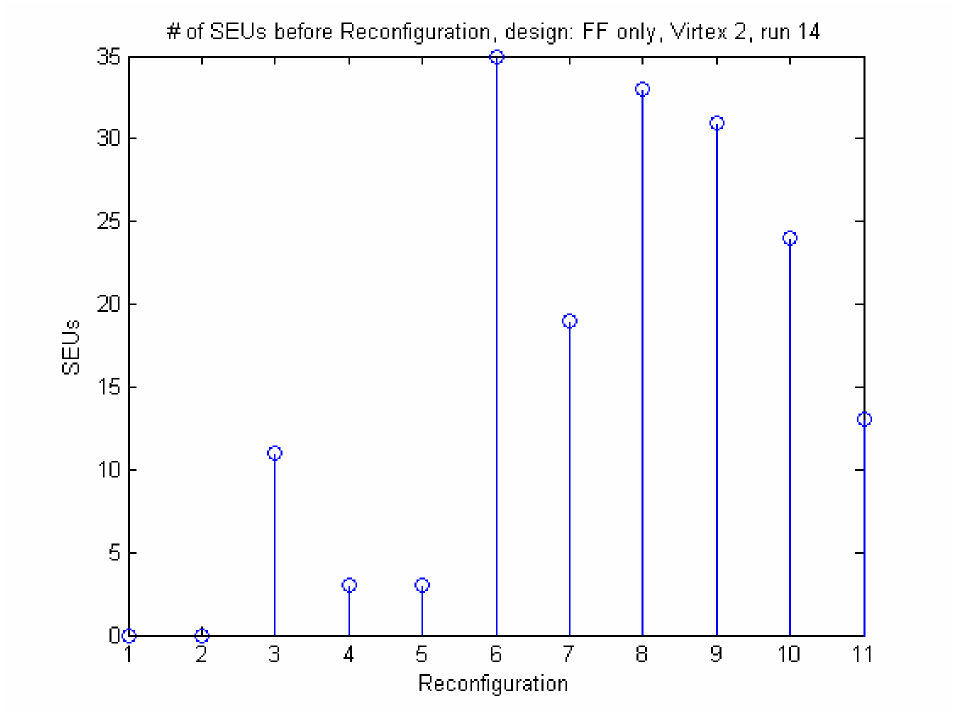


Figure 17. Error-Rate for FF-only (Run Time = 10 minutes)

C. ON-ORBIT SEU-RATE ESTIMATE

The first requirement for estimating on-orbit SEUs is to determine the radiation environment in which CFTP will be operating. CFTP is manifested on NPSAT1, which will orbit at 560 km, 35.4 degrees inclination, and MidSTAR1, which will orbit at 492 km, 46 degrees inclination. Proton environment data was obtained from the CREME96 software, which is based on the National Aeronautics and Space Administration’s (NASA) AP-8 model [32]. Below 1000 km, the average proton-flux is negligible outside the South Atlantic Anomaly (SAA), centered southeast of Brazil at approximately 33 degrees South latitude, 34 degrees West longitude. The NPSAT1 orbit falls in this region, so the estimate was based on using the peak proton flux, while in the SAA.

CREME96 allows the user to select Solar Minimum or Maximum. Solar Minimum was chosen due to the higher resulting proton-flux. CREME96 then displays average flux for 1, 3, 6, 15, 30, 50, and 100 MeV protons, and peak flux for 10, 15, 30, 50 and 100 MeV protons. The displayed flux includes all proton energies at and above that energy level. 30-MeV protons were chosen, because there is no appreciable increase in proton-induced bit-upset cross-section above 30 MeV [2]. The bit cross-section is a measure of susceptibility to SEU in terms of cross-sectional area of the FPGA. Radiation testing for CFTP did not include radiation characterization, but the XQV600 Virtex I is similar to the 300-series Virtex I FPGA (XQVR300) tested by Fuller [2]. So, the Proton SEU Cross Section figure in [2] was used.

The peak proton flux for 560 km (30 MeV protons) is 202 protons/cm^2-sec. Dividing the proton flux of the cyclotron testing (8.78 x 10^5 protons/cm^2-sec) by the peak proton flux on-orbit yields a flux ratio of 4,341. Converting the testing SEU-rate (0.0604 SEU/sec) from seconds to minutes and dividing by flux ratio yields an SEU-rate of 8.35 x 10^{-4} SEU/min. The reader should note that this estimate would be true if NPSAT1 were in the SAA for the entire orbit, so this rate was multiplied by a factor to account for the fraction of the orbit that NPSAT1 is in the SAA. NPSAT1 will fly through the SAA approximately 15 minutes per 96-minute orbit. Over a 24-hour period, NPSAT1 will orbit the earth 15 times, so NPSAT1 will pass through the SAA approximately 225 minutes each day. Multiplying the on-orbit SEU-rate (8.35 x 10^{-4} SEU/min) by 225 min/day yields 0.188 SEU/day, or converserly, <u>an SEU will occur every 5.3 days</u>. The SEU-rate estimates using an average proton-flux for 1000 km and 1500 km (49.9 and 236 protons/cm^2-sec, respectively) are summarized in Table 6.

The SEU-rate for CFTP-1 in NPSAT1 is estimated to be 1 SEU/5-6 days. This estimate does not factor in shielding provided by the spacecraft or aluminum box, so the actual SEU-rate will most likely be lower. If a Virtex II board is flown, the SEU rate will increase by an order of magnitude. Also note the increase in proton flux at higher altitudes. The average proton-flux increases by an order of five just from 1000 km to 1500 km. Higher altitudes would provide an even better environment to test fault-mitigation schemes.

Table 6. On-Orbit SEU-Rate Estimates (SEU/day)

	Peak Flux 560 km	Ave Flux 1000 km	Ave Flux 1500 km
Virtex I	0.188	0.297	1.40
Virtex II	1.76	2.78	13.1

Flux - protons/cm^2-sec

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

A. SUMMARY

The purpose of this thesis was to develop a program for radiation testing of CFTP. This program was designed to establish a proper proton-flux level to obtain SEUs at a rate conducive to follow-on analysis. These results were also used to validate a fault injection tool and estimate the on-orbit SEU-rate for NPSAT1.

Integration and testing methods were introduced, starting with hardware assembly and testing. Software integration with hardware was discussed, along with a brief overview of space qualication of electronic components and how radiation testing is incorporated. The objectives for radiation testing, ground-based and on-orbit, for CFTP were stated.

Radiation test preparations were covered with an explanation of the design philosophy of the shift register. Some valuable lessons-learned were taken from the initial test session in August 2005. One was an equipment issue, where the power-supply current-limit of 1 Amp was too low. This was fixed by using two separate power supplies, with each supply current-limit set at 3 Amps. Streamlining the interface code prior to November testing also helped prevent power supply issues. The key lesson-learned, though, was that the experimenters needed to arrive with fully functional bit-files, ready to test. This was attempted, but not accomplished for the August test-session. Bit-files for five experiments for CFTP-1 (three shift-registers and two CORDIC algorithms) and for six experiments on CFTP-2 (PIX was added to the CFTP-1 slate) were tested prior to arrival at Davis, CA. This hard-work and preparation was reflected in the November test-session, when testing was completed in two days (a day ahead of schedule).

The shift register design was successful in that it utilized the maximum area possible on the FPGA, and provided real-time data for configuration faults that yielded data errors and triggered reconfigurations. This design was verified by introducing a configuration error using FPGA Editor and using a JBits-based fault-injection tool. For CFTP-1, the SRL16-only shift-register appeared to be the most sensitive to faults propogating as data errors. For CFTP-2, though, the flip-flop-only shift-register appeared most sensitive. The differences could be due to the different structure of Virtex 1 and Virtex II FPGAs, but is more likely due to the small statistical sample used for analysis.

The number of faults/SEUs that occurred during testing of CFTP-2 versus CFTP-1 was significantly different. SEUs occurred every 16.6 seconds, on average, for CFTP-1, while SEUs for CFTP-2 occurred every 1.69 seconds, on average. This order of magnitude difference is believed to be due to the smaller feature size and closer packing of the Virtex II FPGA. Continued development and more radiation testing is required to generate more statistically relelvant data. Radiation test results showed a relatively consistent error-rate for different programs, as was expected. This information was then extrapolated using information from CREME96 to estimate the on-orbit SEU-rate for CFTP to be 1 SEU/5-6 days.

Any of the shift-register configurations or CORDIC algorithms for CFTP-1 may be used as experiments on-orbit with NPSAT1 or MidSTAR1. Ideally, though, the PIX processor will be modified and reduced in size to be instantiated onto CFTP-1 for on-orbit testing. This is a good project for future research.

B. FOLLOW-ON RESEARCH

This thesis has only scraped the surface for data collection and analysis for CFTP. Numerous opportunities for future research exist and are required for a more complete guide to the performance of CFTP.

Current programs should be run with the fault injection tool for longer periods to approach asymptotic values (small variances between tests.)

New algorithms need to be developed and tested to explore better fault mitigation techniques.

TMR methods were tested with the CORDIC, and PIX implements a distributed TMR design. Quadded-logic methods could also be tested.

PIX is too large to fit on CFTP-1. Research should be completed to reduce the size of PIX and test this smaller design on CFTP-1. One possible approach to reduce the size is to use a traditional TMR-approach, where the output of three processors is voted, vice the distributed architecture of PIX [7].

Finally, an experiment agenda needs to be developed for implementation on CFTP aboard NPSAT1 and MidSTAR1. An approach similar to the radiation testing developed in this thesis can be used as a template. The goal would be to prepare the same experiments used for radiation testing, or develop new configurations for testing on-orbit.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: CFTP RADIATION TEST PLAN

A. INTRODUCTION

The NPS-led Configurable Fault Tolerant Processor (CFTP) team will conduct proton radiation testing using the Isochronous Cyclotron at Crocker Nuclear Laboratory, University of California-Davis on 14 – 16 November 2005. The devices under test (DUT) will be 2 different development boards based on the CFTP design. The devices are representative of 2 operational units, similar to Naval Postgraduate School's (NPS) CFTP Virtex-I (CFTP-1) design, that are being assembled for integration on two spacecraft that are scheduled to be launched October 12, 2006. The Naval Research Laboratory (NRL) built a board for radiation testing based on the CFTP design with a Virtex-2 FPGA for the experimental FPGA (X2). This test plan outlines facilities, equipment, beam configurations, test descriptions and procedures, agenda and personnel.

B. PURPOSE

The two sample devices will be tested for single event effects (SEE). They will also be monitored for the onset of cumulative (total dose) effects, but the experiments planned should not approach the total dose ratings of either FPGA.. The primary intent of this testing is to assess the Single Event Upset (SEU) susceptibility of the CFTP design (under various configuration loads), determine the SEU-induced fault tolerance, and evaluate partial reconfigurability of tested designs. Test results will also provide a baseline for on-orbit observations of the same experiments. Post-launch modifications to the configuration loads are possible, though more difficult due to bandwidth limitations, so early determination of robust design strategies is beneficial.

C. DEVICES UNDER TEST (DUT)

1. CFTP-1 Development Board

The CFTP-1 development board uses two Xilinx Virtex I, 600,000 gate equivalent, Field Programmable Gate Arrays (FPGAs). This development board was built at Naval Postgraduate School as a prototype for the flight boards and software development model. Highlighted components are seen in Figure 18.

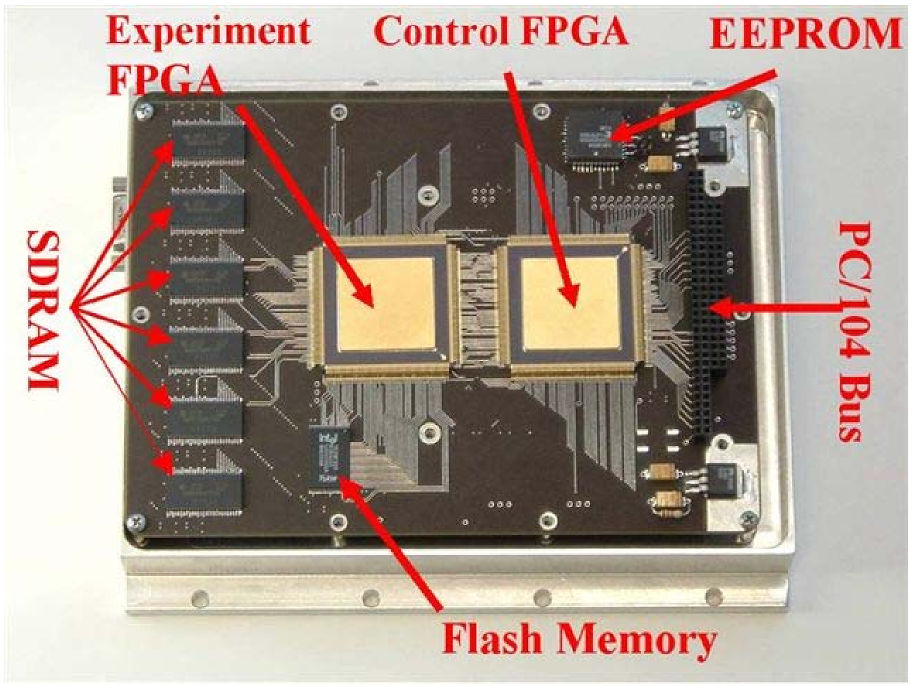


Figure 18. CFTP-1 Layout

One FPGA is used as a configuration controller and is referred to as X1. The second FPGA is the configurable processor, referred to as X2. The focus of this testing will be on X2. The proton beam pattern on X2 is seen in Figure 19.

2. CFTP-2 Development Board

The second development board, pictured in Figure 20. was built by the Naval Research Laboratory and Silver Engineering. This board was built specifically for radiation testing and uses a Xilinx Virtex II, 6 million gate equivalent, FPGA for X2. X1 is a Virtex 1 FPGA, (same as CFTP-1). A Triple Modular Redundant (TMR) MIPS-3000 Processor was originally designed for CFTP-1 to demonstrate the concept of a configurable, fault tolerant processor, but does not fit on the Virtex-I FPGA. This experiment (PIX) will only be performed on the CFTP-2 board. The beam pattern for CFTP-2 is pictured in Figure 20.

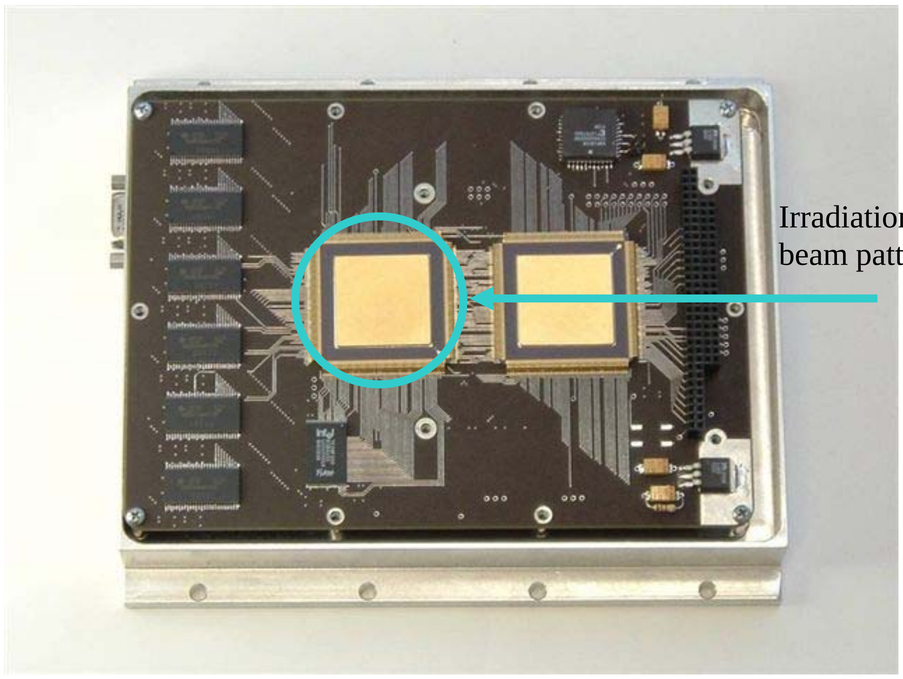


Figure 19. Proton Beam Irradiation Area

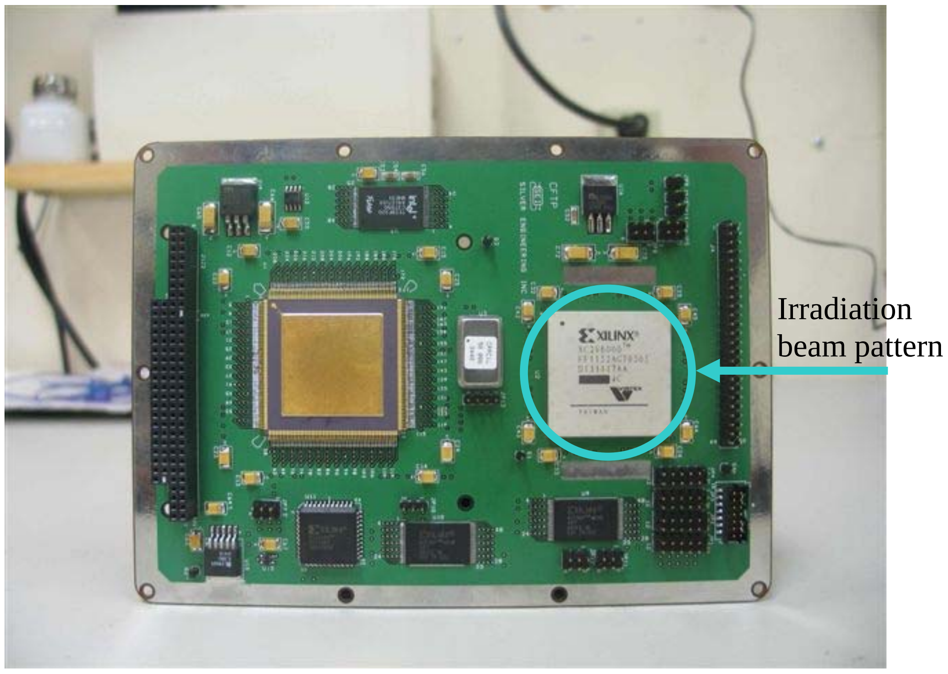


Figure 20. CFTP-2 Layout and Beam Irradiation Area

D. TEST EQUIPMENT

1. UC-Davis Cyclotron

Figure 21. shows the physical layout of the South and North Caves of Crocker Nuclear Laboratory that will be used for this testing. The South Cave will be the experiment control area, where the experimenter's will have connectivity to the Linux laptops controlling the CFTP boards through a Cat 5 Ethernet cable The proton beam entry point is located in the North Cave, where the CFTP boards will be irradiated on the radiation target testing table. The irradiation equipment is a 76 inch cyclotron that provides monoenergetic proton beams up to a maximum energy of 63 MeV. The proton beam has a diameter of 6 cm (2.4 in) and can be further controlled through a Shielding Aperture mounted at the beam output, see Figure 22. This testing will use the 4 cm square aperture to irradiate the entire FPGA, but at the same time, minimize radiation spillover to the rest of the CFTP components. Figure 23. shows an example setup of another experiment completed at the UC-Davis facility, where laser crosshairs are used to ensure proper experiment alignment with the proton beam.

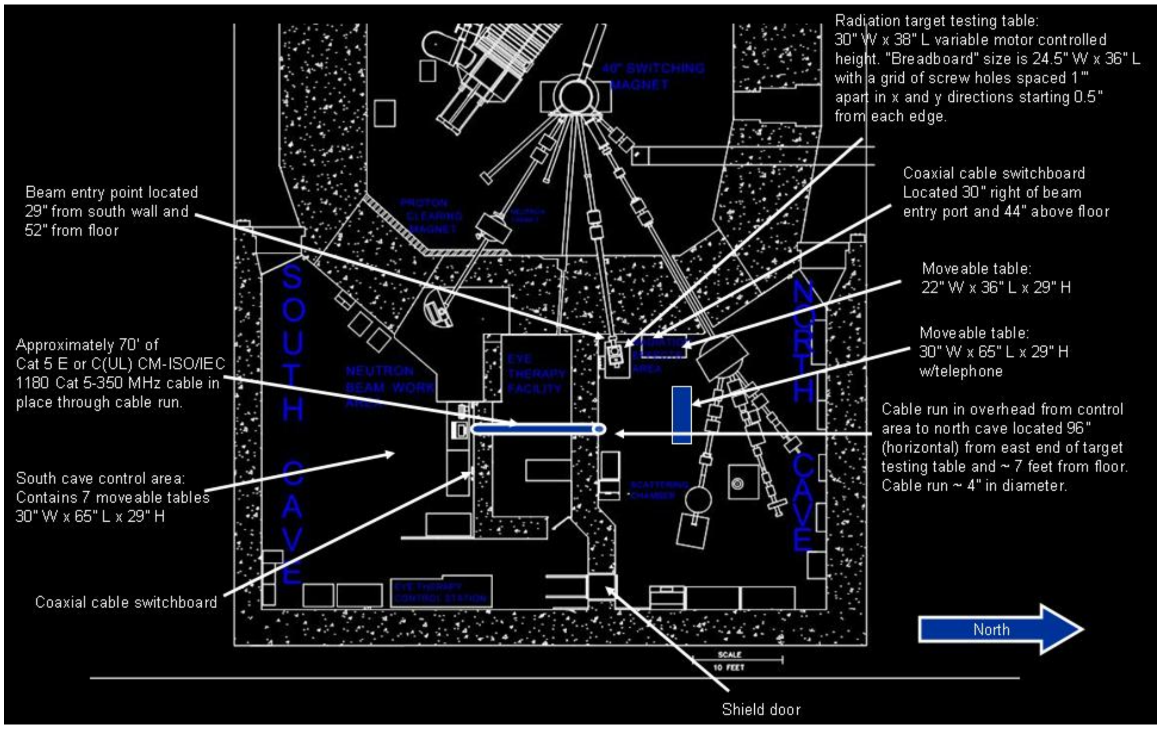

Radiation target testing table: 30" W x 38" L variable motor controlled height. "Breadboard" size is 24.5" W x 36" L with a grid of screw holes spaced 1" apart in x and y directions starting 0.5" from each edge.
40" SWITCHING MAGNET
PROTON CLEARING MAGNET
Beam entry point located 29" from south wall and 52" from floor
Coaxial cable switchboard Located 30" right of beam entry port and 44" above floor
Moveable table: 22" W x 36" L x 29" H
Moveable table: 30" W x 65" L x 29" H w/telephone
SOUTH CAVE
NORTH CAVE
Approximately 70' of Cat 5 E or C(UL) CM-ISO/IEC 1180 Cat 5-350 MHz cable in place through cable run.
NEUTRON BEAM WORK AREA
EYE THERAPY FACILITY
SCATTERING CHAMBER
Cable run in overhead from control area to north cave located 96" (horizontal) from east end of target testing table and ~ 7 feet from floor. Cable run ~ 4" in diameter.
South cave control area: Contains 7 moveable tables 30" W x 65" L x 29" H
Coaxial cable switchboard
EYE THERAPY CONTROL STATION
North
SCALE 10 FEET
Shield door

Figure 21. Crocker Nuclear Laboratory

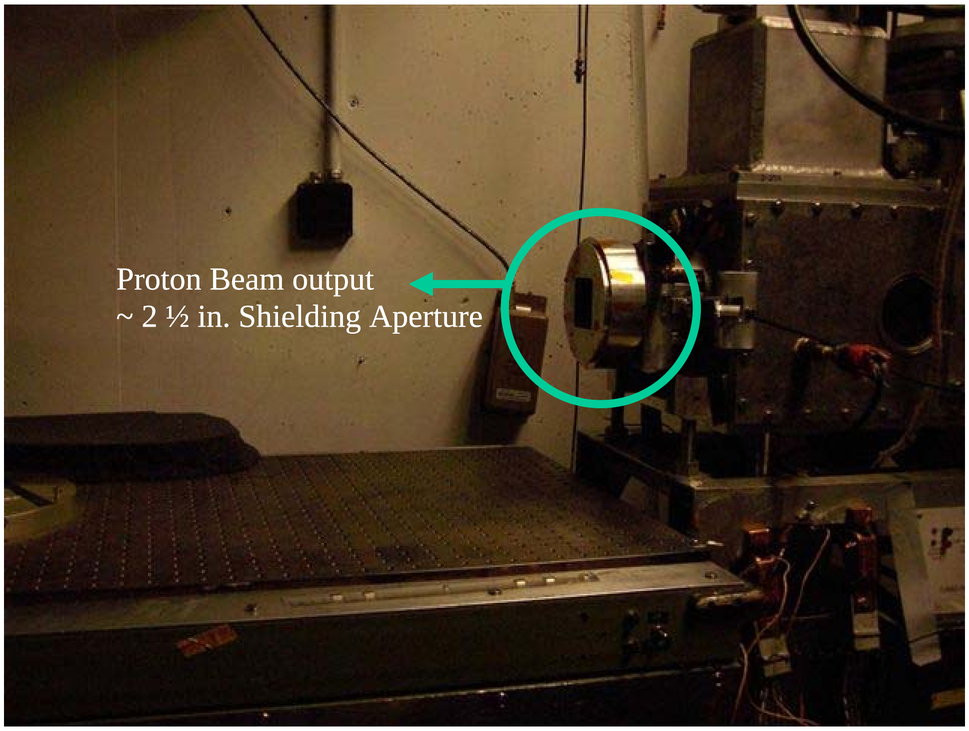


Figure 22. Beam Test Stand with Square Aperture

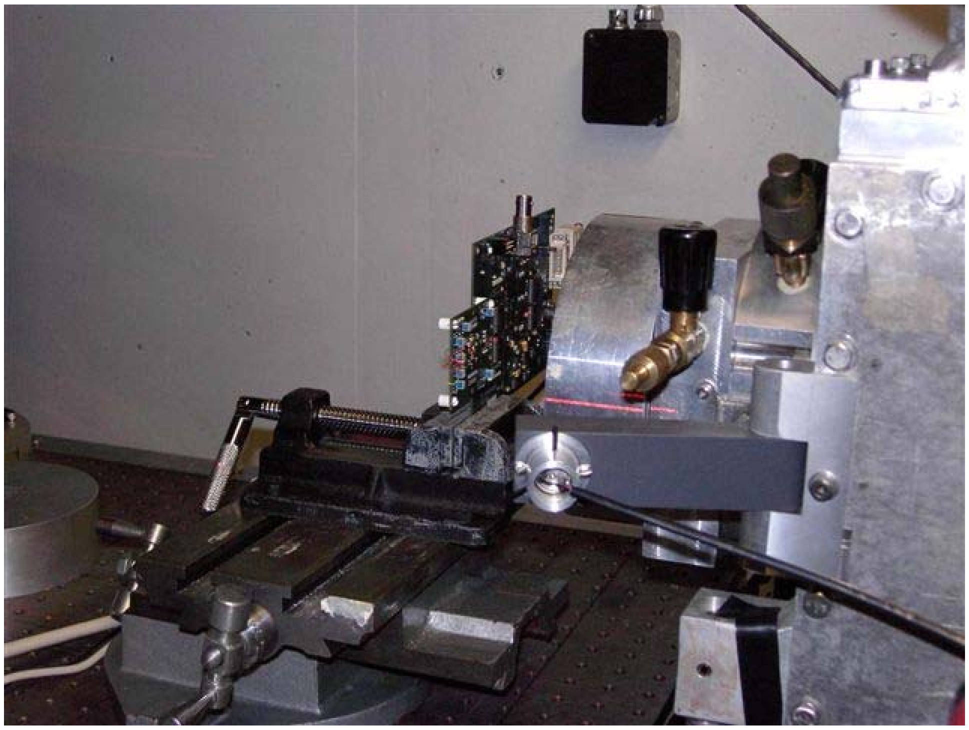


Figure 23. Test Stand for Experiments

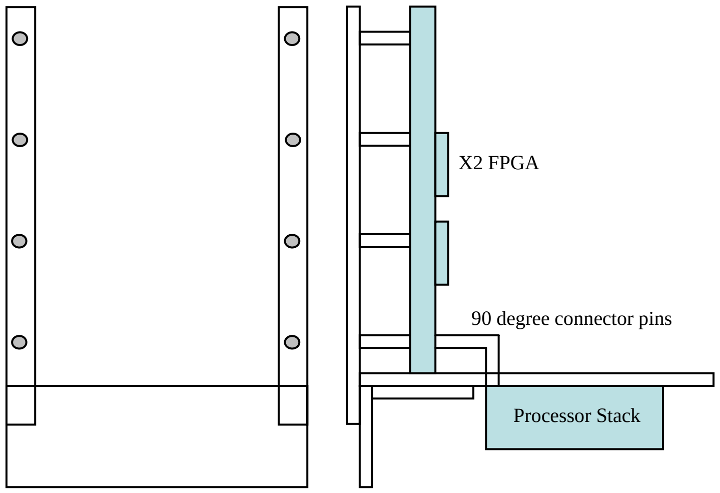
2. Test Configuration (Physical Setup)

a. CFTP-1

Because the CFTP boards are slightly different and were manufactured at different locations, each board has a unique test layout. The test stand for CFTP-1 is seen in Figure 24. A clear plastic plate was placed in front of the board to prevent accidental contact that may damage the board. A square hole was then cut into the plate directly over the X2 FPGA to prevent proton energy dissipation through the plastic. This stand will enable irradiation of the X2 FPGA only. Initial tests will be conducted with X2 orthogonal to the proton beam, because the goal is to maximize SEUs and assess how the configuration responds. Future tests could evaluate the FPGA at different incident angles to the beam using a remotely controlled rotating table that would allow the experiment to rotate while performing a dynamic test.

A Linux-based interface computer will be used in the North Cave (proton beam) to control the experiment. This computer will be connected to the experiment serially to the stack controller and in parallel with the JTAG port. The interface computer will then be connected to the control computer in the South Cave via a 75 foot Ethernet cable. The interface computer will be used to load and configure CFTP via the stack controller. The control computer will be able to start, stop, and reset CFTP; receive and log data from CFTP; and display in real time the SEU count, last error condition code, and other diagnostics. The list of hardware and software are summarized in Table 7 and Table 8, respectively.

Two power supplies will be used; the first power supply will provide 5 volts (from the 6 V source). This will allow experimenters to monitor the current to the CFTP board itself. This power supply (Agilent E3631A) will be controlled by the control computer via a serial connection (RS-232). Current monitoring will be accomplished using the Microsoft Excel plug-in provided by Agilent Technologies. The second power supply will provide power to the stack controller and hard drive. Both power supplies will plug into a remote power controller to enable experimenters to shut down the experiment via the Ethernet connection, if necessary. The cabling layout for CFTP-1 is pictured in Figure 25.



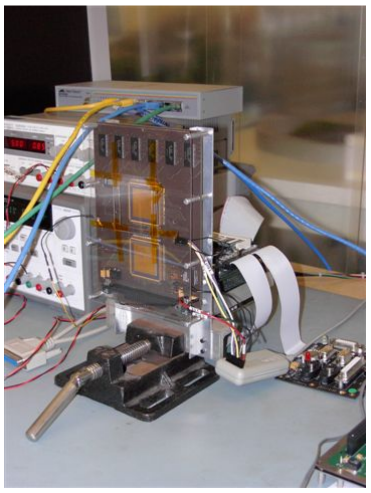


Figure 24. CFTP-1 Test Stand

Table 7. Equipment List

Equipment	**Purpose**	**Box**
CFTP-1 Experiment Vice	 Mount experiment	
Power Supply #1 (E3631A) Power cord 60 ft serial cable	Power CFTP-1 Board Control E3631A power supply	
Power Supply (HP ???) Power cord	Power CFTP-1 Stack controller	
Power Controller	Remote shutdown capability	
CFTP Interface Laptop Power cord Ethernet cable Xilinx parallel cable 10 ft serial cable	Serial ppp connection to CFTP Stack Controller Connection to hub in North Cave Parallel JTAG connection to CFTP Board. Serial connection to CFTP-1	
Monitor Keyboard	Monitor CFTP experiments during setup	
Control Laptop (Dell D810) Power cord Ethernet cable	Control experiment from outside vault. Connection to hub in South Cave	
Control Laptop (Dell D610) Power cord Ethernet cable	Control power supply; log current. Connection to hub in South Cave	
Ethernet hub (D-Link) Power cable	South Cave	
Ethernet hub (???) Power cable	North Cave	
Webcam Power cable Ethernet cable	Monitor current Connection to hub in North Cave	
CFTP-2 Experiment		
Power Supply #2 (E3631A) Power cord	Power CFTP-2 Board Note: use 60 ft serial cable listed above	
Power Supply (???)	Power CFTP-2 Backplane and hard drive	
CFTP Interface Laptop (Power cord Ethernet cable Xilinx parallel cable 10 ft serial cable	Serial ppp connection to CFTP Stack Controller Connection to hub in North Cave Parallel JTAG connection to CFTP Board. Serial connection to CFTP-2	
XX Spare Ethernet cables		
Power strip	Power laptops/experiments in South Cave	
Digital Camera		
2 Walkie Talkies	Communicate between North and South Caves	

Table 8. Software List

Control Laptop	**Interface Laptop**
Windows XP OS	Linux OS
MS Office (Excel for current logging)	Xilinx tools Impact XDL
Putty (Secure Shell program)	Configuration files
Xilinx ISE	JBits 2.8
Agilent E3631A Excel plug-in	
CDs	
Backup of configuration files	

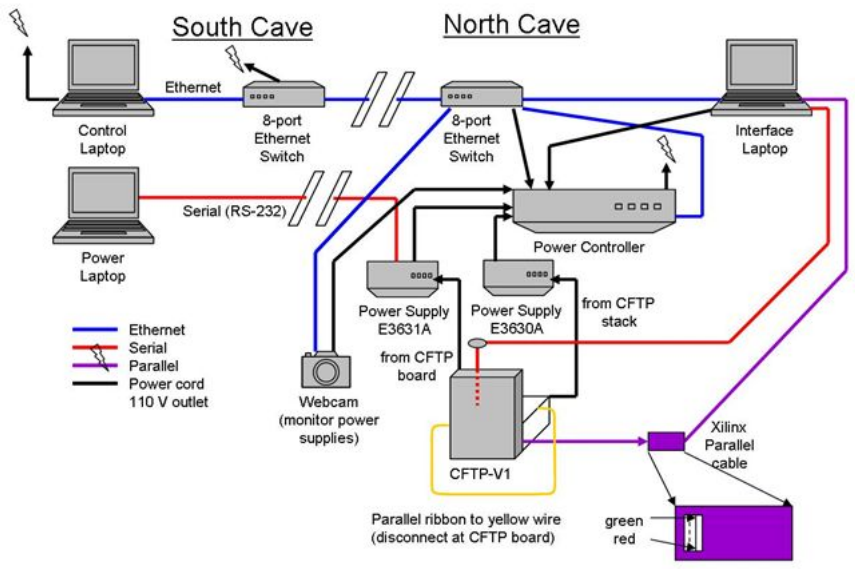


Figure 25. CFTP-1 Cabling Layout

b. CFTP-2

CFTP-2 is a stand-alone experiment that is operated concurrently with CFTP-1 at NPS. This set-up will be used at Davis to minimize equipment deviations during testing and allow for work on either CFTP board while the other is being irradiated. CFTP-2 will utilize a TMZ104 Stack controller and a TRI-M Engineering DEV104-ISA backplane for testing. Power is normally supplied to CFTP-2 with one power supply; however, current monitoring to the CFTP board could not be achieved. For radiation testing, an additional PC/104 connector was added between the backplane and CFTP board with the 5 V pin clipped and routed to a separate power supply. This second power supply will be another Agilent E3631A, which enables current monitoring using the same process as that for CFTP-1. The cabling layout for CFTP-2 is similar to that of CFTP-1, with differences seen in Figure 26.

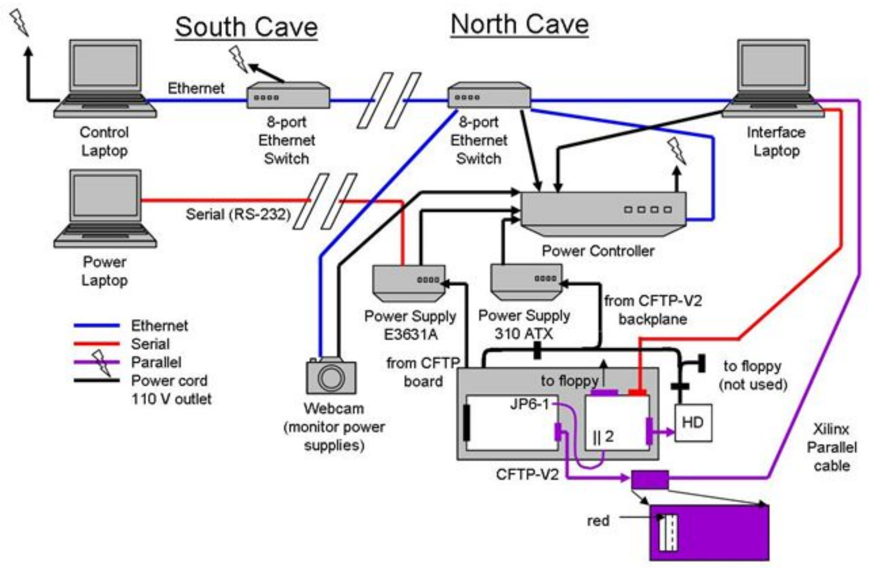


Figure 26. CFTP-2 Cabling Layout

E. TEST DESCRIPTIONS/PROCEDURES

These tests will evaluate current programs/configurations for SEU tolerance and partial reconfiguration and provide experimental data to enhance configurations for future radiation testing and on-orbit operation. Care will be taken to mitigate total dose radiation damage to either experiment, which will be accomplished by limiting the total fluence level to less than 1 kRAD for each experiment. A summary of planned experiments are shown in Table 9.

Table 9. Experiment Summary

Experiment	**Purpose**
Beam Exposure	Verify proper aperture is being used for FPGAs.
Shift Register (CFTP-1)	Verify expected data and configuration SEUs and map the configuration errors.
CORDIC-Approximate (CFTP-1)	Evaluate algorithm.
CORDIC-TMR (CFTP-1)	Evaluate algorithm.
Shift Register (CFTP-2)	Verify expected data and configuration SEUs and map the configuration errors.
CORDIC-Approximate (CFTP-2)	Evaluate algorithm.
CORDIC-TMR (CFTP-2)	Evaluate algorithm.
PIX (CFTP-2)	Evaluate processor.
Additional configurations (CFTP-1 or CFTP-2)	Evaluate alternate configurations

The power-up procedure detailed in Table 10 will be followed for all experiments.

Table 10. Power-Up Procedure

1. Turn on E3631A Power Supply (*Do not connect the CFTP experiment.*) Using the Excel plug-in on the serially connected laptop, Set the Voltage Limit on the 6 V source to ***5 Volts*** Set the Current Limit on the 6 V source to ***3 Amps***
2. (For CFTP-1) Turn on the E3630A Power Supply Set the Voltage to ***5 Volts*** (*Be careful not to subsequently bump the voltage dial.*)
3. Connect the CFTP board to the E3631A Power Supply
4. (For CFTP-1) Connect the Stack Controller to the E3630A Power Supply (For CFTP-2) Connect the Backplane and Hard drive to the 310 ATX Power Supply
5. Start current log and chart on laptop.
6. Simultaneously, turn E3631A Output ON (using laptop) and turn on other power supply (use walkie talkies)
7. Monitor current for anomalies during start-up

1. Beam Exposure

Obtain a beam exposure using the 4 cm square aperture. Overlay the exposure on top of the X2 FPGA on both CFTP boards to verify sufficient radiation coverage. Verify multiple blank data sheets (Appendix A-2) are available in the experimenter’s area in the South Cave.

2. Shift Register (CFTP-1)

Set-up CFTP-1 on the radiation target testing table. Connect all cables, except power to the CFTP board and stack controller, according to Figure 26. Use the network information displayed in the South Cave to set up the IP connections for the control and interface computers, web cam, and power controller. Record IP settings using data sheet in Appendix A-1. Then, follow the power-up procedure in Table 10. Continually monitor current for any increase, terminating the test for any increase more than 10% (possible indication of total dose effects). Using Putty (or other secure shell client) on the control computer, connect to the interface computer. Load the shift register experiment into the flash. Perform a flash dump and checkflash to confirm the flash is properly loaded. Then, execute the shift register program. After observing proper operation of the shift register program, close the North Cave. Log the experiment number, run and start time on a data sheet.

Set up the beam for 63 MeV protons and start with 5 picoAmps for the desired fluence. Irradiate the experiment and log the time. Observe the program output for SEUs and reconfiguration. Log the time for any data errors and when reconfigurations occur. The objective is to observe one SEU approximately every 30 seconds. If SEU occurrence is more frequent, stop the beam and lower the current (cyclotron control). Start the experiment over, logging information on a new data sheet. Once the beam is set at the proper fluence, run the shift register experiment until approximately 100 SEUs are observed. This should yield a dose of 300 rad. Ensure that the total dose does not exceed 500 rad for this experiment. Note: CFTP-1 has already received 191 rad of proton irradiation.

After achieving the desired number of SEUs, stop the beam and log the time. Verify the output is saved to a file for follow-on analysis. Using the SelectMap outputs from 30 second updates and reconfigurations, determine the expected SEUs (data vs. configuration and location {components or routing}) for the CORDIC experiments.

3. CORDIC Approximate (CFTP-1)

When ready to continue, load the CORDIC approximate experiment into the flash. Perform a flash dump and checkflash to confirm the flash is properly loaded. Then, execute the CORDIC approximate program. After observing proper operation of the program, close the North Cave. Log the experiment number, run and start time on a data sheet. Continually monitor current for any increase, terminating the test for any increase more than 10%.

Set up the beam for the last setting used for the shift register experiment. Irradiate the experiment and log the time. Observe the program output for SEUs and reconfiguration. Log the time for any data errors and when reconfigurations occur. If SEU occurrence is too frequent, stop the beam and lower the current (cyclotron control). Start the experiment over, logging information on a new data sheet. Run the shift register experiment until approximately 100 SEUs are observed. This should yield a dose of 300 rad. Ensure that the total dose does not exceed 1 krad for this experiment.

After achieving the desired number of SEUs, stop the beam and log the time. Verify the output is saved to a file for follow-on analysis. Note the cumulative dose for CFTP-1 to date. Using the SelectMap outputs from 30 second updates and reconfigurations, analyze the observed SEUs (data vs. configuration and location {components or routing}).

4. CORDIC TMR (CFTP-1)

When ready to continue, load the CORDIC TMR experiment into the flash. Perform a flash dump and checkflash to confirm the flash is properly loaded. Then, execute the CORDIC TMR program. After observing proper operation of the program, close the North Cave. Log the experiment number, run and start time on a data sheet. Continually monitor current for any increase, terminating the test for any increase more than 10%.

Set up the beam for the last setting used for the CORDIC approximate experiment. Irradiate the experiment and log the time. Observe the program output for SEUs and reconfiguration. Log the time for any data errors and when reconfigurations occur. If SEU occurrence is too frequent, stop the beam and lower the current (cyclotron control). Start the experiment over, logging information on a new data sheet. Run the shift register experiment until approximately 100 SEUs are observed. This should yield a dose of 300 rad. Ensure that the total dose does not exceed 1 krad for this experiment.

After achieving the desired number of SEUs, stop the beam and log the time. Verify the output is saved to a file for follow-on analysis. Note the cumulative dose for CFTP-1 to date. Using the SelectMap outputs from 30 second updates and reconfigurations, analyze the observed SEUs (data vs. configuration and location {components or routing}).

5. Shift Register (CFTP-2)

Set-up CFTP-2 on the radiation target testing table. Connect all cables, except power to the CFTP board and stack controller, according to Figure 26. Verify proper network set-up. Then, follow the power-up procedure in Table 10. Continually monitor current for any increase, terminating the test for any increase more than 10% (possible indication of total dose effects). Using Putty (or other secure shell client) on the control computer, connect to the interface computer. Load the shift register experiment (for CFTP-2) into the flash. Perform a flash dump and checkflash to confirm the flash is properly loaded. Then, execute the shift register program. After observing proper operation of the shift register program, close the North Cave. Log the experiment number, run and start time on a data sheet.

Set up the beam for the last setting used for the CORDIC TMR experiment. Irradiate the experiment and log the time. Observe the program output for SEUs and reconfiguration. Log the time for any data errors and when reconfigurations occur. NOTE: Because the Virtex II FPGA is more dense (10 times the equivalent gates), it may be more sensitive to SEUs. The objective is to observe one SEU approximately every 30 seconds. If SEU occurrence is more frequent, stop the beam and lower the current (cyclotron control). Start the experiment over, logging information on a new data sheet. Once the beam is set at the proper fluence, run the shift register experiment until approximately 100 SEUs are observed. This should yield a dose of 300 rad. Ensure that the total dose does not exceed 500 rad for this experiment. Note: This is the first time CFTP-2 has been irradiated.

After achieving the desired number of SEUs, stop the beam and log the time. Verify the output is saved to a file for follow-on analysis. Using the SelectMap outputs from 30 second updates and reconfigurations, determine the expected SEUs (data vs. configuration and location {components or routing}) for the CORDIC experiments.

6. CORDIC Approximate (CFTP-2)

When ready to continue, load the CORDIC approximate (for CFTP-2) experiment into the flash. Perform a flash dump and checkflash to confirm the flash is properly loaded. Then, execute the CORDIC approximate program. After observing proper operation of the program, close the North Cave. Log the experiment number, run and start time on a data sheet. Continually monitor current for any increase, terminating the test for any increase more than 10%.

Set up the beam for the last setting used for the shift register experiment. Irradiate the experiment and log the time. Observe the program output for SEUs and reconfiguration. Log the time for any data errors and when reconfigurations occur. If SEU occurrence is too frequent, stop the beam and lower the current (cyclotron control). Start the experiment over, logging information on a new data sheet. Run the shift register experiment until approximately 100 SEUs are observed. This should yield a dose of 300 rad. Ensure that the total dose does not exceed 1 krad for this experiment.

After achieving the desired number of SEUs, stop the beam and log the time. Verify the output is saved to a file for follow-on analysis. Note the cumulative dose for CFTP-1 to date. Using the SelectMap outputs from 30 second updates and reconfigurations, analyze the observed SEUs (data vs. configuration and location {components or routing}).

7. CORDIC TMR (CFTP-2)

When ready to continue, load the CORDIC TMR (for CFTP-2) experiment into the flash. Perform a flash dump and checkflash to confirm the flash is properly loaded. Then, execute the CORDIC TMR program. After observing proper operation of the program, close the North Cave. Log the experiment number, run and start time on a data sheet. Continually monitor current for any increase, terminating the test for any increase more than 10%.

Set up the beam for the last setting used for the shift register experiment. Irradiate the experiment and log the time. Observe the program output for SEUs and reconfiguration. Log the time for any data errors and when reconfigurations occur. If SEU occurrence is too frequent, stop the beam and lower the current (cyclotron control).

Start the experiment over, logging information on a new data sheet. Run the shift register experiment until approximately 100 SEUs are observed. This should yield a dose of 300 rad. Ensure that the total dose does not exceed 1 krad for this experiment.

After achieving the desired number of SEUs, stop the beam and log the time. Verify the output is saved to a file for follow-on analysis. Note the cumulative dose for CFTP-1 to date. Using the SelectMap outputs from 30 second updates and reconfigurations, analyze the observed SEUs (data vs. configuration and location {components or routing}).

8. PIX (CFTP-2)

When ready to continue, load the CORDIC TMR (for CFTP-2) experiment into the flash. Perform a flash dump and checkflash to confirm the flash is properly loaded. Then, execute the CORDIC TMR program. After observing proper operation of the program, close the North Cave. Log the experiment number, run and start time on a data sheet. Continually monitor current for any increase, terminating the test for any increase more than 10%.

Set up the beam for the last setting used for the shift register experiment. Irradiate the experiment and log the time. Observe the program output for SEUs and reconfiguration. Log the time for any data errors and when reconfigurations occur. If SEU occurrence is too frequent, stop the beam and lower the current (cyclotron control). Start the experiment over, logging information on a new data sheet. Run the shift register experiment until approximately 100 SEUs are observed. This should yield a dose of 300 rad. Ensure that the total dose does not exceed 1 krad for this experiment.

After achieving the desired number of SEUs, stop the beam and log the time. Verify the output is saved to a file for follow-on analysis. Note the cumulative dose for CFTP-1 to date. Using the SelectMap outputs from 30 second updates and reconfigurations, analyze the observed SEUs (data vs. configuration and location {components or routing}).

9. Additional Testing

Perform additional tests with new configurations or new programs for above configurations as time allows.

F. TEST AGENDA

Table 11 contains the agenda for testing.

Table 11. Test Agenda

Time	Mon 14 Nov	Tue 15 Nov	Wed 16 Nov	Thu 17 Nov
0800-1600	Personnel Arrive Set-up CFTP-1 in North Cave	Data Analysis Revise Test Plan Set-up CFTP-2 in North Cave	Data Analysis Test Setup	Personnel Depart
1600-2400	Test Shift Register Test CORDIC Set-up CFTP-2 Test Shift Register	Test CORDIC Test PIX Additional Testing	Additional Testing Breakdown/Pack-up	

G. SHIPPING INFORMATION:

Test equipment should be sent to:

DR PAUL MARSHALL
CARE OF MR CARLOS CASTANEDA
CROCKER NUCLEAR LABORATORY, UC DAVIS
ONE SHIELDS AVENUE
DAVIS CA 95616-8569
(530) 752-1460 or (530)-752-4228

H. TEST SITE PERSONNEL

1. Test Director/Radiation Engineer

Paul Marshall (NASA-GSFC)
Email: PWMarshall@aol.com

2. On-Site NPS personnel

Professor Herschel Loomis
Email: hloomis@nps.edu
Professor Alan Ross
Email: aross@nps.edu
LCDR James Coudeyras
Email: jccoudey@nps.edu
LT Pete Majewicz
Email: pmajewic@nps.edu
Tim Meehan
Email: tjmeehan@nps.edu
Capt Josh Snodgrass
Email: jdsnodgr@nps.edu
Mindy Surratt
Email: mlsurrat@nps.edu

3. On-site NRL personnel

Doug Disabello
Email: douglas.disabello@gmail.com
Kjell Tengesdal
Email: tengesdal2@llnl.gov

4. Off-site NPS personnel (phone support, etc.)

David Rigmaiden
Email: drigmaiden@nps.edu

5. Off-site additional personnel (phone support, etc.)

John Willis
Email: john.willis@ftlsys.com

6. UC Davis Test Facility

Test Area Telephone: 530-754-9289

Facility POC:
Carlos Castaneda
Email: Castaneda@Crocker.UCDavis.Edu

APPENDIX B: LAN IP ADDRESS ASSIGNMENTS

Table 12. IP Assignment Log

Davis Facility	IP Addresses
Usable Static IP Addresses	169.237.209.150-169.237.209.170
Subnet Mask	255.255.255.192
Gateway/Router Address	169.237.209.190
DNS Addresses	169.237.1.250 169.237.250.250

Component	Assigned IP Address (see Usable Static IP Addresses above)
Control Laptop	-
CFTP-***1*** Interface Laptop	-
CFTP-***2*** Interface Laptop	-
Webcam	-
Power Controller	-
Power Laptop	-
Additional Assignments	
1. Mindy Laptop	-
2.	-
3.	-
4.	-
5.	-
6.	-
7.	-
8.	-
9.	-
10.	-

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: RADIATION TEST DATA SHEET

Table 13. Experiment Description

Experiment Number (see Table): ____________

Run Number: ____________

Experiment Times (Use 24-hour clock)

Start Experiment ____________
Start Radiation ____________
Stop Radiation ____________

Experiment Description	Exp #
CFTP-V1 Shift Register (SRL)	1
CFTP-V1 CORDIC (approx)	2
CFTP-V1 CORDIC (TMR)	3
CFTP-V1 SRL+1	4
CFTP-V1 No SRL/FF only	5
CFTP-V2 Shift Register (SRL)	6
CFTP-V2 CORDIC (approx)	7
CFTP-V2 CORDIC (TMR)	8
CFTP-V2 PIX	9
CFTP-V2 SRL+1	10
CFTP-V2 No SRL/FF only	11

	1	2	3	4	5
Data Error					
▨					
▨					
▨					
▨					
▨					
▨					
▨					
▨					
▨					
Reconfiguration					

	6	7	8	9	10
Data Error					
▨					
▨					
▨					
▨					
▨					
▨					
▨					
▨					
▨					
Reconfiguration					

NOTES:

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: CODE

Appendix D contains the VHDL code for the all shift register programs and the top level code for X1 and all sub-modules. MATLAB code used for data analysis is also listed in this appendix.

A. SHIFT REGISTER MODULE WITH SRL16E MACRO

```
-------------------------------------------------
-- filename: sr_testing.vhd
-- author: James Coudeyras (2005)
--
-- This file is the basic shift register module
-- used for X2 which will be used as the initial test
-- for proton radiation testing at UC-Davis.
--
-------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sr_testing is

    generic ( WIDTH : integer := 2400); -- SR length (2400 for CFTP-1)

    Port  ( clock       : in std_logic;
        reset           : in std_logic;
        ce              : in std_logic;
        din             : in std_logic;
        dout            : out std_logic;
        sumdiff         : out std_logic );

end sr_testing;

architecture sr_sequence of sr_testing is
    signal reg_a    : std_logic_vector (WIDTH-1 downto 0);
    signal reg_b    : std_logic_vector (WIDTH-1 downto 0);

begin

process (clock, reset)
begin
    if (reset = '1') then

        sumdiff <= '0';
        dout <=  '0';

    elsif (clock'event and clock='1') then

        sumdiff <= '0';

        if ce='1' then
            reg_a <= din & reg_a(WIDTH-1 downto 1);
            reg_b <= din & reg_b(WIDTH-1 downto 1);
        end if;
-- LABEL1:
        for I in 1 to (WIDTH/16) loop
            if ( (reg_a((I-1)*16) xor reg_b((I-1)*16)) = '1' ) then
                sumdiff <= '1';
            end if;
        end loop;
        dout <= reg_a(0);
    end if;
end process;
end sr_sequence;
```

B. X2 TOP LEVEL CODE

```
-------------------------------------------------
-- filename: cftp_x2_sr.vhd
-- author: James Coudeyras (2005)
--
-- This file is the top level code for X2 to generate sixteen columns
-- of the shift register and define the interface signals with X1.
-- This top level code will be used for X2 as the initial test
-- for proton radiation testing at UC-Davis.
--
-- This code is the same for all versions of the shift register.
-------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity cftp_x2 is
    Port (
        clock               : in std_logic;
        T_CE_FROM_X1_i      : in std_logic;
        T_RESET_FROM_X1_i   : in std_logic;
        T_BITIN_FROM_X1_i   : in std_logic;
        T_DOUT_TO_X1_o      : out std_logic; -- verifying LFSR output
        T_XOR_DOUT_TO_X1_o  : out std_logic; -- comparing douts on X2
        T_SUMDIFF_TO_X1_o   : out std_logic_vector(15 downto 0));
end cftp_x2;

architecture sr_columns of cftp_x2 is

    component sr_testing port (
        clock   : in std_logic;
        reset   : in std_logic;
        ce      : in std_logic; -- clock enable
        din     : in std_logic; -- data bit in from LFSR
        dout    : out std_logic; -- data bit out to counter
        sumdiff : out std_logic := '0'); -- don't need for X1
    end component;

--REMOVE FOR SIMULATION
    component BUFG port (
        I                   : in std_logic;
        O                   : out std_logic);
    end component;

    signal s_clock_X2_i     : std_logic; -- 12.5 MHz system clock w/
            reset
    signal s_clock_X2       : std_logic; -- 12.5 MHz system clock w/
            reset
    signal s_clock_X2_cnt   : integer range 0 to 16;

    signal s_dout_o         : std_logic_vector(15 downto 0);

begin

-- Divide 25 MHz clock to 12.5 MHz to meet timing constraint
```

```
    process (clock, T_RESET_FROM_X1_i) begin
        if (T_RESET_FROM_X1_i = '1') then
            s_clock_x2_i <= '0';
        elsif (clock'event and clock = '1') then
            if (s_clock_X2_cnt >= 1) then  -- this achieves a divide by
                2 clock period!!!
                s_clock_X2_i <= not s_clock_X2_i;
                s_clock_X2_cnt <= 0;
            else
                s_clock_X2_cnt <= s_clock_X2_cnt + 1;
            end if;
        end if;
    end process;
-- FOR SR implementation {END}


    gen: for i in 15 downto 0 generate
        U: entity sr_testing
-- verify broken line does not cause problems
         port map (clock => s_clock_X2, reset => T_RESET_FROM_X1_i, ce
                => T_CE_FROM_X1_i, din => T_BITIN_FROM_X1_i, dout =>
                s_dout_o(i), sumdiff => T_SUMDIFF_TO_X1_o(i));
    end generate;

    process (s_clock_X2) begin
        if (clock'event and clock = '1') then
            T_DOUT_TO_X1_o <= s_dout_o(0);
            T_XOR_DOUT_TO_X1_o <= (s_dout_o(0) xor s_dout_o(1) xor
                s_dout_o(2) xor s_dout_o(3) xor s_dout_o(4) xor
                s_dout_o(5) xor s_dout_o(6) xor s_dout_o(7) xor
                s_dout_o(8) xor s_dout_o(9) xor s_dout_o(10) xor
                s_dout_o(11) xor s_dout_o(12) xor s_dout_o(13) xor
                s_dout_o(14) xor s_dout_o(15)); -- '0';
        end if;
    end process;

--Clock distribution network for X2 clock
    s_clock_X2_bufg : BUFG port map ( -- comment out this code only for
                simulations!!!
        I           => s_clock_X2_i,
        O           => s_clock_x2
    );

end sr_columns;
```

C. X2 SHIFT REGISTER CONSTRAINT FILE (VIRTEX I)

```
# Pin assignments for X2
#
# double-check all pin assignments???
# these numbers derived from a Mar 2004 diagram
#

# system clock, pin 87 on X2
NET "clock" LOC = "P87";
# NET "CLOCK" PERIOD = 40;  # Why specify the clock period???
# NET "s_clock" PERIOD = 80;
# the line above gave errors during "Translate"???

# signals to/from X1
NET "T_BITIN_FROM_X1_i" LOC = "p132";  # X1_X2_AUX<0>
NET "T_CE_FROM_X1_i" LOC = "p134";  # X1_X2_AUX<1>
NET "T_RESET_FROM_X1_i" LOC = "p135";  # X1_X2_AUX<2>
# NET "XXX" LOC = "p136";  # X1_X2_AUX<3>
# NET "XXX" LOC = "p138";  # X1_X2_AUX<4>
# NET "XXX" LOC = "p139";  # X1_X2_AUX<5>
# NET "XXX" LOC = "p141";  # X1_X2_AUX<6>
# NET "XXX" LOC = "p144";  # X1_X2_AUX<7>
# NET "XXX" LOC = "p146";  # X1_X2_AUX<8>
# NET "XXX" LOC = "p147";  # X1_X2_AUX<9>
NET "T_SUMDIFF_TO_X1_o<0>" LOC = "p153";  # X1_X2_AUX<10>
NET "T_DOUT_TO_X1_o" LOC = "p154";  # X1_X2_AUX<11>
NET "T_SUMDIFF_TO_X1_o<1>" LOC = "p159";  # X1_X2_AUX<12>
NET "T_XOR_DOUT_TO_X1_o" LOC = "p160";  # X1_X2_AUX<13>
NET "T_SUMDIFF_TO_X1_o<2>" LOC = "p161";  # X1_X2_AUX<14>
#NET "T_DOUT_TO_X1_o<2>" LOC = "p177";  # X1_X2_AUX<15>
NET "T_SUMDIFF_TO_X1_o<3>" LOC = "p178";  # X1_X2_AUX<16>
#NET "T_DOUT_TO_X1_o<3>" LOC = "p179";  # X1_X2_AUX<17>
NET "T_SUMDIFF_TO_X1_o<4>" LOC = "p181";  # X1_X2_AUX<18>
#NET "T_DOUT_TO_X1_o<4>" LOC = "p182";  # X1_X2_AUX<19>
NET "T_SUMDIFF_TO_X1_o<5>" LOC = "p183";  # X1_X2_AUX<20>
#NET "T_DOUT_TO_X1_o<5>" LOC = "p184";  # X1_X2_AUX<21>
NET "T_SUMDIFF_TO_X1_o<6>" LOC = "p185";  # X1_X2_AUX<22>
#NET "T_DOUT_TO_X1_o<6>" LOC = "p188";  # X1_X2_AUX<23>
NET "T_SUMDIFF_TO_X1_o<7>" LOC = "p189";  # X1_X2_AUX<24>
#NET "T_DOUT_TO_X1_o<7>" LOC = "p190";  # X1_X2_AUX<25>
NET "T_SUMDIFF_TO_X1_o<8>" LOC = "p192";  # X1_X2_AUX<26>
#NET "T_DOUT_TO_X1_o<8>" LOC = "p193";  # X1_X2_AUX<27>
NET "T_SUMDIFF_TO_X1_o<9>" LOC = "p194";  # X1_X2_AUX<28>
#NET "T_DOUT_TO_X1_o<9>" LOC = "p195";  # X1_X2_AUX<29>
NET "T_SUMDIFF_TO_X1_o<10>" LOC = "p196";  # X1_X2_AUX<30>
#NET "T_DOUT_TO_X1_o<10>" LOC = "p197";  # X1_X2_AUX<31>
NET "T_SUMDIFF_TO_X1_o<11>" LOC = "p198";  # X1_X2_AUX<32>
#NET "T_DOUT_TO_X1_o<11>" LOC = "p204";  # X1_X2_AUX<33>
NET "T_SUMDIFF_TO_X1_o<12>" LOC = "p205";  # X1_X2_AUX<34>
#NET "T_DOUT_TO_X1_o<12>" LOC = "p206";  # X1_X2_AUX<35>
NET "T_SUMDIFF_TO_X1_o<13>" LOC = "p207";  # X1_X2_AUX<36>
#NET "T_DOUT_TO_X1_o<13>" LOC = "p208";  # X1_X2_AUX<37>
NET "T_SUMDIFF_TO_X1_o<14>" LOC = "p209";  # X1_X2_AUX<38>
#NET "T_DOUT_TO_X1_o<14>" LOC = "p211";  # X1_X2_AUX<39>
NET "T_SUMDIFF_TO_X1_o<15>" LOC = "p212";  # X1_X2_AUX<40>
#NET "T_DOUT_TO_X1_o<15>" LOC = "p213";  # X1_X2_AUX<41>
```

```
# NET "XXX" LOC = "p216";  # X1_X2_AUX<42>
# NET "XXX" LOC = "p217";  # X1_X2_AUX<43>
# NET "XXX" LOC = "p218";  # X1_X2_AUX<44>

# Start of Constraints extracted by Floorplanner from the Design
INST "U15_*" AREA_GROUP = "cftp_x2_sr_15" ;
AREA_GROUP "cftp_x2_sr_15" RANGE = CLB_R1C1:CLB_R48C4 ;
INST "U14_*" AREA_GROUP = "cftp_x2_sr_14" ;
AREA_GROUP "cftp_x2_sr_14" RANGE = CLB_R1C6:CLB_R48C9 ;
INST "U13_*" AREA_GROUP = "cftp_x2_sr_13" ;
AREA_GROUP "cftp_x2_sr_13" RANGE = CLB_R1C10:CLB_R48C13 ;
INST "U12_*" AREA_GROUP = "cftp_x2_sr_12" ;
AREA_GROUP "cftp_x2_sr_12" RANGE = CLB_R1C15:CLB_R48C18 ;
INST "U11_*" AREA_GROUP = "cftp_x2_sr_11" ;
AREA_GROUP "cftp_x2_sr_11" RANGE = CLB_R1C19:CLB_R48C22 ;
INST "U10_*" AREA_GROUP = "cftp_x2_sr_10" ;
AREA_GROUP "cftp_x2_sr_10" RANGE = CLB_R1C24:CLB_R48C27 ;
INST "U9_*" AREA_GROUP = "cftp_x2_sr_9" ;
AREA_GROUP "cftp_x2_sr_9" RANGE = CLB_R1C28:CLB_R48C31 ;
INST "U8_*" AREA_GROUP = "cftp_x2_sr_8" ;
AREA_GROUP "cftp_x2_sr_8" RANGE = CLB_R1C33:CLB_R48C36 ;
INST "U7_*" AREA_GROUP = "cftp_x2_sr_7" ;
AREA_GROUP "cftp_x2_sr_7" RANGE = CLB_R1C37:CLB_R48C40 ;
INST "U6_*" AREA_GROUP = "cftp_x2_sr_6" ;
AREA_GROUP "cftp_x2_sr_6" RANGE = CLB_R1C42:CLB_R48C45 ;
INST "U5_*" AREA_GROUP = "cftp_x2_sr_5" ;
AREA_GROUP "cftp_x2_sr_5" RANGE = CLB_R1C46:CLB_R48C49 ;
INST "U4_*" AREA_GROUP = "cftp_x2_sr_4" ;
AREA_GROUP "cftp_x2_sr_4" RANGE = CLB_R1C51:CLB_R48C54 ;
INST "U3_*" AREA_GROUP = "cftp_x2_sr_3" ;
AREA_GROUP "cftp_x2_sr_3" RANGE = CLB_R1C55:CLB_R48C58 ;
INST "U2_*" AREA_GROUP = "cftp_x2_sr_2" ;
AREA_GROUP "cftp_x2_sr_2" RANGE = CLB_R1C60:CLB_R48C63 ;
INST "U1_*" AREA_GROUP = "cftp_x2_sr_1" ;
AREA_GROUP "cftp_x2_sr_1" RANGE = CLB_R1C64:CLB_R48C67 ;
INST "U0_*" AREA_GROUP = "cftp_x2_sr_0" ;
AREA_GROUP "cftp_x2_sr_0" RANGE = CLB_R1C69:CLB_R48C72 ;
# AREA_GROUP "sr_sample_2" RANGE = RAMB4_R0C0:RAMB4_R4C0 ;  # No RAMB4
                used here!!!
```

D. X2 SHIFT REGISTER CONSTRAINT FILE (VIRTEX II)

```
# Pin assignments for X2
#
# double-check all pin assignments???
# these numbers derived from a Mar 2004 diagram
#

# system clock, pin 87 on X2
NET "clock" LOC = "P87";
# NET "CLOCK" PERIOD = 40;  # Why specify the clock period???
# NET "s_clock" PERIOD = 80;
# the line above gave errors during "Translate"???

# signals to/from X1
NET "T_BITIN_FROM_X1_i" LOC = "p132";  # X1_X2_AUX<0>
NET "T_CE_FROM_X1_i" LOC = "p134";  # X1_X2_AUX<1>
NET "T_RESET_FROM_X1_i" LOC = "p135";  # X1_X2_AUX<2>
# NET "XXX" LOC = "p136";  # X1_X2_AUX<3>
# NET "XXX" LOC = "p138";  # X1_X2_AUX<4>
# NET "XXX" LOC = "p139";  # X1_X2_AUX<5>
# NET "XXX" LOC = "p141";  # X1_X2_AUX<6>
# NET "XXX" LOC = "p144";  # X1_X2_AUX<7>
# NET "XXX" LOC = "p146";  # X1_X2_AUX<8>
# NET "XXX" LOC = "p147";  # X1_X2_AUX<9>
NET "T_SUMDIFF_TO_X1_o<0>" LOC = "p153";  # X1_X2_AUX<10>
NET "T_DOUT_TO_X1_o" LOC = "p154";  # X1_X2_AUX<11>
NET "T_SUMDIFF_TO_X1_o<1>" LOC = "p159";  # X1_X2_AUX<12>
NET "T_XOR_DOUT_TO_X1_o" LOC = "p160";  # X1_X2_AUX<13>
NET "T_SUMDIFF_TO_X1_o<2>" LOC = "p161";  # X1_X2_AUX<14>
#NET "T_DOUT_TO_X1_o<2>" LOC = "p177";  # X1_X2_AUX<15>
NET "T_SUMDIFF_TO_X1_o<3>" LOC = "p178";  # X1_X2_AUX<16>
#NET "T_DOUT_TO_X1_o<3>" LOC = "p179";  # X1_X2_AUX<17>
NET "T_SUMDIFF_TO_X1_o<4>" LOC = "p181";  # X1_X2_AUX<18>
#NET "T_DOUT_TO_X1_o<4>" LOC = "p182";  # X1_X2_AUX<19>
NET "T_SUMDIFF_TO_X1_o<5>" LOC = "p183";  # X1_X2_AUX<20>
#NET "T_DOUT_TO_X1_o<5>" LOC = "p184";  # X1_X2_AUX<21>
NET "T_SUMDIFF_TO_X1_o<6>" LOC = "p185";  # X1_X2_AUX<22>
#NET "T_DOUT_TO_X1_o<6>" LOC = "p188";  # X1_X2_AUX<23>
NET "T_SUMDIFF_TO_X1_o<7>" LOC = "p189";  # X1_X2_AUX<24>
#NET "T_DOUT_TO_X1_o<7>" LOC = "p190";  # X1_X2_AUX<25>
NET "T_SUMDIFF_TO_X1_o<8>" LOC = "p192";  # X1_X2_AUX<26>
#NET "T_DOUT_TO_X1_o<8>" LOC = "p193";  # X1_X2_AUX<27>
NET "T_SUMDIFF_TO_X1_o<9>" LOC = "p194";  # X1_X2_AUX<28>
#NET "T_DOUT_TO_X1_o<9>" LOC = "p195";  # X1_X2_AUX<29>
NET "T_SUMDIFF_TO_X1_o<10>" LOC = "p196";  # X1_X2_AUX<30>
#NET "T_DOUT_TO_X1_o<10>" LOC = "p197";  # X1_X2_AUX<31>
NET "T_SUMDIFF_TO_X1_o<11>" LOC = "p198";  # X1_X2_AUX<32>
#NET "T_DOUT_TO_X1_o<11>" LOC = "p204";  # X1_X2_AUX<33>
NET "T_SUMDIFF_TO_X1_o<12>" LOC = "p205";  # X1_X2_AUX<34>
#NET "T_DOUT_TO_X1_o<12>" LOC = "p206";  # X1_X2_AUX<35>
NET "T_SUMDIFF_TO_X1_o<13>" LOC = "p207";  # X1_X2_AUX<36>
#NET "T_DOUT_TO_X1_o<13>" LOC = "p208";  # X1_X2_AUX<37>
NET "T_SUMDIFF_TO_X1_o<14>" LOC = "p209";  # X1_X2_AUX<38>
#NET "T_DOUT_TO_X1_o<14>" LOC = "p211";  # X1_X2_AUX<39>
NET "T_SUMDIFF_TO_X1_o<15>" LOC = "p212";  # X1_X2_AUX<40>
#NET "T_DOUT_TO_X1_o<15>" LOC = "p213";  # X1_X2_AUX<41>
```

```
# NET "XXX" LOC = "p216";  # X1_X2_AUX<42>
# NET "XXX" LOC = "p217";  # X1_X2_AUX<43>
# NET "XXX" LOC = "p218";  # X1_X2_AUX<44>

# Start of Constraints extracted by Floorplanner from the Design
INST "U15_*" AREA_GROUP = "cftp_x2_sr_15" ;
AREA_GROUP "cftp_x2_sr_15" RANGE = CLB_R1C1:CLB_R48C4 ;
INST "U14_*" AREA_GROUP = "cftp_x2_sr_14" ;
AREA_GROUP "cftp_x2_sr_14" RANGE = CLB_R1C6:CLB_R48C9 ;
INST "U13_*" AREA_GROUP = "cftp_x2_sr_13" ;
AREA_GROUP "cftp_x2_sr_13" RANGE = CLB_R1C10:CLB_R48C13 ;
INST "U12_*" AREA_GROUP = "cftp_x2_sr_12" ;
AREA_GROUP "cftp_x2_sr_12" RANGE = CLB_R1C15:CLB_R48C18 ;
INST "U11_*" AREA_GROUP = "cftp_x2_sr_11" ;
AREA_GROUP "cftp_x2_sr_11" RANGE = CLB_R1C19:CLB_R48C22 ;
INST "U10_*" AREA_GROUP = "cftp_x2_sr_10" ;
AREA_GROUP "cftp_x2_sr_10" RANGE = CLB_R1C24:CLB_R48C27 ;
INST "U9_*" AREA_GROUP = "cftp_x2_sr_9" ;
AREA_GROUP "cftp_x2_sr_9" RANGE = CLB_R1C28:CLB_R48C31 ;
INST "U8_*" AREA_GROUP = "cftp_x2_sr_8" ;
AREA_GROUP "cftp_x2_sr_8" RANGE = CLB_R1C33:CLB_R48C36 ;
INST "U7_*" AREA_GROUP = "cftp_x2_sr_7" ;
AREA_GROUP "cftp_x2_sr_7" RANGE = CLB_R1C37:CLB_R48C40 ;
INST "U6_*" AREA_GROUP = "cftp_x2_sr_6" ;
AREA_GROUP "cftp_x2_sr_6" RANGE = CLB_R1C42:CLB_R48C45 ;
INST "U5_*" AREA_GROUP = "cftp_x2_sr_5" ;
AREA_GROUP "cftp_x2_sr_5" RANGE = CLB_R1C46:CLB_R48C49 ;
INST "U4_*" AREA_GROUP = "cftp_x2_sr_4" ;
AREA_GROUP "cftp_x2_sr_4" RANGE = CLB_R1C51:CLB_R48C54 ;
INST "U3_*" AREA_GROUP = "cftp_x2_sr_3" ;
AREA_GROUP "cftp_x2_sr_3" RANGE = CLB_R1C55:CLB_R48C58 ;
INST "U2_*" AREA_GROUP = "cftp_x2_sr_2" ;
AREA_GROUP "cftp_x2_sr_2" RANGE = CLB_R1C60:CLB_R48C63 ;
INST "U1_*" AREA_GROUP = "cftp_x2_sr_1" ;
AREA_GROUP "cftp_x2_sr_1" RANGE = CLB_R1C64:CLB_R48C67 ;
INST "U0_*" AREA_GROUP = "cftp_x2_sr_0" ;
AREA_GROUP "cftp_x2_sr_0" RANGE = CLB_R1C69:CLB_R48C72 ;
# AREA_GROUP "sr_sample_2" RANGE = RAMB4_R0C0:RAMB4_R4C0 ;  # No RAMB4
                used here!!!
```

E. X1 SELECTMAP CONFIGURATION CODE

(From [17])

```
-----------------------------------------------------------------------
-- selectmap_config.vhd                                              --
-- Author: Mindy Surratt, 2005                                       --
-- Research Associate, Naval Postgraduate School, Monterey, CA       --
--                                                                   --
-- Code to perform a selectmap full configuration on the V1          --
-- experiment FPGA                                                   --
-----------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity selectmap_config is

    port (

        T_CLOCK_i           : in std_logic;
        RESET_i             : in std_logic;

        T_SELECTMAP_INIT_o  : out std_logic;
        T_SELECTMAP_WRITE_o : out std_logic;
        T_SELECTMAP_CS_o    : out std_logic;
        T_SELECTMAP_DATA_o  : out std_logic_vector(7 downto 0);

        SM_CONFIG_RQST_i    : in std_logic;
        SM_CONFIG_STATUS_o  : out std_logic;

        T_FLASH_DATA_i      : in std_logic_vector(15 downto 0);
        T_FLASH_ADDRESS_o   : out std_logic_vector(21 downto 0);

        PC104_WR_RDY_i      : in std_logic;
        PC104_WR_EN_o       : out std_logic;
        DATA_o              : out std_logic_vector(7 downto 0)

    );

end selectmap_config;

architecture rtl of selectmap_config is

CONSTANT BIN_LENGTH             : integer := 450996; --SIMULATION 20;
CONSTANT CONFIG_DELAY           : integer := 4000; --SIMULATION 10;
CONSTANT ABORT_SETUP_LENGTH     : integer := 5;
CONSTANT ABORT_LENGTH           : integer := 5;
CONSTANT ABORT_RELEASE_LENGTH   : integer := 5;
CONSTANT CONFIG_LENGTH          : integer := CONFIG_DELAY + BIN_LENGTH
                +
                                                ABORT_SETUP_LENGTH +
                                                ABORT_LENGTH +
                                                ABORT_RELEASE_LENGTH;
```

```
signal count_config         : integer range 0 to CONFIG_LENGTH ;
signal byte_count           : integer range 0 to 100;
signal byte1                : std_logic;
signal byte2                : std_logic;
signal s_flash_address_o    : std_logic_vector(20 downto 0);

begin

-- only one flash device on CFTP-1, so only 21 addr lines (addr(21) =
                '0')
T_FLASH_ADDRESS_o(21) <= '0';
T_FLASH_ADDRESS_o(20 downto 0) <= s_flash_address_o;

-- Selectmap Configuration Process
process(T_CLOCK_i, RESET_i)
begin
                if (RESET_i = '1') then

        s_flash_address_o <= "000000000000000000000";

        SM_CONFIG_STATUS_o <= '0';

        T_SELECTMAP_DATA_o <= x"00";

                  T_SELECTMAP_INIT_o <= '0';
                  T_SELECTMAP_WRITE_o <= '1';
                  T_SELECTMAP_CS_o <= '1';

        byte1 <= '0';
        byte2 <= '0';
        PC104_WR_EN_o <= '0';

                  count_config <= CONFIG_LENGTH;

                elsif(T_CLOCK_i'event and T_CLOCK_i = '1') then

        T_SELECTMAP_INIT_o <= '1';
        T_SELECTMAP_CS_o <= '1';
        T_SELECTMAP_WRITE_o <= '1';
        PC104_WR_EN_o <= '0';

        if ( count_config = 0 ) then
            if (PC104_WR_RDY_i = '1') then
                -- Write 'S' to PC104
                if (byte1 = '0') then
                    DATA_o <= x"53";
                    PC104_WR_EN_o <= '1';
                    byte1 <= '1';
                -- Write 'C' to PC104
                else
                    DATA_o <= x"43";
                    PC104_WR_EN_o <= '1';
                    byte2 <= '1';
                end if;
            end if;
            if (byte2 = '1') then
                count_config <= 1;
                byte1 <= '0';
                byte2 <= '0';
            end if;

        -- give it some time
        elsif (count_config < CONFIG_DELAY ) then
            count_config <= count_config + 1;

        -- assert write and CS
        elsif (count_config < CONFIG_DELAY + ABORT_SETUP_LENGTH) then
            count_config <= count_config + 1;
            T_SELECTMAP_CS_o <= '0';
            T_SELECTMAP_WRITE_o <= '0';

        -- deassert write while CS is asserted (pulls an abort)
        elsif (count_config < CONFIG_DELAY + ABORT_SETUP_LENGTH +
                ABORT_LENGTH) then
            count_config <= count_config + 1;
            T_SELECTMAP_CS_o <= '0';
            T_SELECTMAP_WRITE_o <= '1';

        -- Just deassert WRITE and CS to release the abort (for Virtex
                1)
        -- (Default values of WRITE and CS are '1', see above)
        elsif (count_config < CONFIG_DELAY + ABORT_SETUP_LENGTH +
                                ABORT_LENGTH + ABORT_RELEASE_LENGTH)
                then
            count_config <= count_config + 1;

        -- read data from flash and write to X2s selectmap interface
        elsif (count_config < CONFIG_LENGTH) then
            byte_count <= byte_count + 1;

            --T_SELECTMAP_DATA_io0 is the MSB (except I swapped values
                in UCF
            --so in this instance D7 is MSB...)
            -- disabled bit flipping in promgen (default is to flip
                bits in
            -- each byte), so T_SELECTMAP_DATA_o<=t_flash_data instead
                of
            -- having to reverse the bits (7=0, 6=1, etc)

            -- takes some time to read from the flash (possibly less
                than
            -- I'm allowing for...
            if (byte_count = 6) then
                T_SELECTMAP_CS_o <= '0';
                T_SELECTMAP_WRITE_o <= '0';
                T_SELECTMAP_DATA_o <= T_FLASH_DATA_i(7 downto 0);
                count_config <= count_config + 1;
                byte_count <= 0;
                s_flash_address_o <= s_flash_address_o + 1;
            end if;

        -- sit here until we get a config request from the top level
        elsif (count_config = CONFIG_LENGTH ) then
```

```
            count_config <= CONFIG_LENGTH;
            SM_CONFIG_STATUS_o <= '0';
            --reset flash address
            s_flash_address_o <= "000000000000000000000";
            if (SM_CONFIG_RQST_i = '1') then
                count_config <= 0;
               SM_CONFIG_STATUS_o <= '1';
            end if;

        end if;


    end if;

end process;

end rtl;
```

F. X1 SELECTMAP READBACK CODE

```
----------------------------------------------------------------------
-- selectmap_readback.vhd                                           --
-- Author: Mindy Surratt, 2005                                      --
-- Research Associate, Naval Postgraduate School, Monterey, CA      --
--                                                                  --
-- Code to perform a selectmap readback on the experiment FPGA,     --
-- compare the readback data with the configuration file and mask   --
-- file stored in the Flash memory, and output any configuration    --
-- errors to the PC/104                                             --
----------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity selectmap_readback is
    port (

        T_CLOCK_i           : in std_logic;
        CLOCK_i             : in std_logic;
        CLOCK_NOBUFG_i      : in std_logic;
        RESET_i             : in std_logic;

        T_CCLK_o            : out std_logic;
        T_SELECTMAP_INIT_o  : out std_logic;
        T_SELECTMAP_WRITE_o : out std_logic;
        T_SELECTMAP_CS_o    : out std_logic;
        T_SELECTMAP_DATA_o  : out std_logic_vector(7 downto 0);
        T_SELECTMAP_DATA_i  : in std_logic_vector(7 downto 0);

        SM_RB_RQST_i        : in std_logic;
        SM_RB_STATUS_o      : out std_logic;

        T_FLASH_DATA_i      : in std_logic_vector(15 downto 0);
        T_FLASH_ADDRESS_o   : out std_logic_vector(21 downto 0);

        PC104_WR_RDY_i      : in std_logic;
        PC104_WR_EN_o       : out std_logic;
        DATA_o              : out std_logic_vector(7 downto 0)

    );
end selectmap_readback;

architecture rtl of selectmap_readback is

-- Constants

    CONSTANT READBACK_COMMAND_LENGTH    : integer := 36;
    -- CLB_LENGTH refers to CLBs, IOBs, and BRAMIs. We do not read back
            BRAMs
    CONSTANT CLB_LENGTH                 : integer := 435240; --
            SIMULATION 130;
    CONSTANT READBACK_DELAY             : integer := 4000; --SIMULATION
            10;
    CONSTANT READBACK_INIT1_DELAY       : integer := 1;
    CONSTANT READBACK_INIT2_DELAY       : integer := 1; -- DO NOT
            CHANGE THIS!
    -- Number of clocks needed to set up readback (before we actually
            start
    -- getting data)
    CONSTANT READBACK_OFFSET            : integer :=
            READBACK_COMMAND_LENGTH +
                                                    READBACK_DELAY
            +

            READBACK_INIT1_DELAY +

            READBACK_INIT2_DELAY;
    CONSTANT READBACK_LENGTH            : integer := CLB_LENGTH +

            READBACK_OFFSET;

    signal s_clock_i        : std_logic;

-- Flash compare/report signals

    type   my_state is (sleep_st,read_flash_st,wr_hdr_1_st,wr_hdr_2_st,
                    read_sm_st, wait_st, compare_st,
            write_pc104_st);
    type   error_vector is array(15 downto 0) of std_logic_vector(7
            downto 0);

    signal report_error_state       : my_state;
    signal error_word               : error_vector;

    signal flash_data_reg           : std_logic_vector(15 downto 0);
    signal s_flash_address_o        : std_logic_vector(20 downto 0);
    signal error_count              : integer range 0 to 1024;
    signal error_location           : std_logic_vector (23 downto 0);
    signal error_location_readback  : std_logic_vector (23 downto 0);
    signal wr_cnt                   : integer range 0 to 32;


-- Selectmap Readback signals

    type   readback_mem_type is array(READBACK_COMMAND_LENGTH-1 downto
            0)
                            of std_logic_vector(7 downto 0);

    signal readback_command     : readback_mem_type;
    signal count_readback       : integer range 0 to 1024000;
    signal strt_rb              : std_logic;
    signal s_selectmap_data_d   : std_logic_vector(7 downto 0);
    signal s_selectmap_data_i   : std_logic_vector(7 downto 0);
    signal reading_sm_data      : std_logic;
    signal reading_sm_data_d    : std_logic;
    signal readback_location    : integer range 0 to 1024000;

    signal selectmap_read_data  : std_logic_vector(7 downto 0);
```

```
begin

    -- Clocking CCLK only works if we use a signal that is not on the
    -- clock network
    process (T_CLOCK_i) begin
        if (T_CLOCK_i'event and T_CLOCK_i = '1') then
            s_clock_i <= not s_clock_i;
        end if;
    end process;


    -- Command sequence that initializes a selectmap readback
    readback_command(0)(7 downto 0) <= x"FF"; --MSB dummy word
    readback_command(1)(7 downto 0) <= x"FF";
    readback_command(2)(7 downto 0) <= x"FF";
    readback_command(3)(7 downto 0) <= x"FF";
    readback_command(4)(7 downto 0) <= x"AA"; --MSB synchronization
                word
    readback_command(5)(7 downto 0) <= x"99";
    readback_command(6)(7 downto 0) <= x"55";
    readback_command(7)(7 downto 0) <= x"66";
    readback_command(8)(7 downto 0) <= x"30"; --MSB write 1 word to FAR
                reg
    readback_command(9)(7 downto 0) <= x"00";
    readback_command(10)(7 downto 0) <= x"20";
    readback_command(11)(7 downto 0) <= x"01";
    readback_command(12)(7 downto 0) <= x"00"; --MSB FAR address
    readback_command(13)(7 downto 0) <= x"00";
    readback_command(14)(7 downto 0) <= x"00";
    readback_command(15)(7 downto 0) <= x"00";
    readback_command(16)(7 downto 0) <= x"30"; --MSB write 1 word to
                CMD reg
    readback_command(17)(7 downto 0) <= x"00";
    readback_command(18)(7 downto 0) <= x"80";
    readback_command(19)(7 downto 0) <= x"01";
    readback_command(20)(7 downto 0) <= x"00"; --MSB RCFG command
    readback_command(21)(7 downto 0) <= x"00";
    readback_command(22)(7 downto 0) <= x"00";
    readback_command(23)(7 downto 0) <= x"04";
    readback_command(24)(7 downto 0) <= x"28"; --MSB type 2 header
    readback_command(25)(7 downto 0) <= x"00";
    readback_command(26)(7 downto 0) <= x"60";
    readback_command(27)(7 downto 0) <= x"00";
    readback_command(28)(7 downto 0) <= x"48"; --type 2 read from
                active reg
    readback_command(29)(7 downto 0) <= x"01"; --(FDRO) 0x1A90A 32 bit
                words
    readback_command(30)(7 downto 0) <= x"A9"; -- (READBACK_LENGTH/4)
    readback_command(31)(7 downto 0) <= x"0A";
    readback_command(32)(7 downto 0) <= x"00"; --MSB pad word
    readback_command(33)(7 downto 0) <= x"00";
    readback_command(34)(7 downto 0) <= x"00";
    readback_command(35)(7 downto 0) <= x"00";
```

```
-- grabs selectmap data on the next read_sm_st,
-- the first byte read back will be junk (no data yet)
T_CCLK_o            <= '1' when reading_sm_data = '1'
                                 and report_error_state /=
            read_sm_st
                            else s_clock_i;
s_selectmap_data_i  <= T_SELECTMAP_DATA_i;

-- only one flash device on CFTP-1, so only 21 addr lines (addr(21)
            = '0')
T_FLASH_ADDRESS_o(21) <= '0';
T_FLASH_ADDRESS_o(20 downto 0) <= s_flash_address_o;

readback_location <= 0  when count_readback < READBACK_OFFSET
                        else (count_readback - READBACK_OFFSET);

process(CLOCK_i,RESET_i)
begin
    if (RESET_i = '1') then

        report_error_state <= sleep_st;
        error_count <= 0;
        wr_cnt <= 0;

        PC104_WR_EN_o <= '0';
        DATA_o <= x"31";

    elsif (CLOCK_i'event and CLOCK_i = '1') then

        PC104_WR_EN_o <= '0';
        report_error_state <= report_error_state;

        case report_error_state is

            when sleep_st =>

                -- set flash address to just after the selectmap
                -- configuration commands in the bin file
                s_flash_address_o <= "0000000000000001001000";
                -- wait until we start actually reading SM data
                -- then start the readback reports
                if (reading_sm_data = '1') then
                    report_error_state <= wr_hdr_1_st;
                end if;

            when wr_hdr_1_st =>

                -- Write 'S' to PC104
                if (PC104_WR_RDY_i='1' ) then
                    DATA_o <= x"53"; --S
                    PC104_WR_EN_o <= '1';
                    report_error_state <= wr_hdr_2_st;
                end if;

            when wr_hdr_2_st =>

                -- Write 'M' to PC104
                if (PC104_WR_RDY_i='1' ) then
                    DATA_o <= x"4D"; --M
                    PC104_WR_EN_o <= '1';
                    report_error_state <= read_flash_st;
                end if;

            when read_flash_st =>

                report_error_state <= read_sm_st;

                -- Go idle if we are done reading back
                if (reading_sm_data = '0') then
                    report_error_state <= sleep_st;
                end if;

            when read_sm_st =>

                -- latch the selectmap data from the last read,
                -- and trigger CCLK to get the next byte (see
            T_CCLK_o
                -- assignment above)
                selectmap_read_data <= s_selectmap_data_i;
                report_error_state <= wait_st;

            when wait_st =>

                -- Skip first junk byte (see CCLK above), 1 dummy
            word,
                -- and one pad frame (120 bytes)
                if (readback_location < 125) then
                    report_error_state <= read_sm_st;
                else
                    report_error_state <= compare_st;
                end if;

            when compare_st =>

                -- latch flash data for possible report to PC104
                flash_data_reg <= T_FLASH_DATA_i;
                -- increment the flash address for the next read
                s_flash_address_o <= s_flash_address_o + 1;

                -- Compare read back selectmap data, configuration
            data
                -- stored in the flash, and mask data stored in the
            flash.
                -- If there is a mismatch, print a report to
            PC/104.
                -- Otherwise, read the next byte
                if ( ( (T_FLASH_DATA_i(7 downto 0) xor
            selectmap_read_data)
                    and not T_FLASH_DATA_i(15 downto 8) ) /= x"00"
            ) then
                    report_error_state <= write_pc104_st;
                else
                    report_error_state <= read_flash_st;
                end if;
```

```
            when write_pc104_st =>

                -- print out SM report
                -- error_word() defined below
                if (wr_cnt < 7 and PC104_WR_RDY_i = '1' ) then
                    DATA_o <= error_word(wr_cnt);
                    PC104_WR_EN_o <= '1';
                    wr_cnt <= wr_cnt + 1;
                -- read next byte from selectmap when done
                elsif (wr_cnt = 7) then
                    report_error_state <= read_flash_st;
                    wr_cnt <= 0;
                end if;

            when others => report_error_state <= sleep_st;
        end case;
    end if;

end process;

error_location <= "000" & (s_flash_address_o - 1) ;
error_location_readback <=
        std_logic_vector(to_unsigned(readback_location,24));

-- SMRB error report, error location is byte location within
-- bin file
error_word(0) <= x"00";
error_word(1) <= selectmap_read_data ;
error_word(2) <= error_location(23 downto 16);
error_word(3) <= error_location(15 downto 8);
error_word(4) <= error_location(7 downto 0);
error_word(5) <= flash_data_reg(7 downto 0);
error_word(6) <= flash_data_reg(15 downto 8);


-- Process to perform Selectmap Readback
process(CLOCK_i,RESET_i)
begin
    if (RESET_i = '1') then

        T_SELECTMAP_INIT_o <= '1';
        T_SELECTMAP_WRITE_o <= '1';
        T_SELECTMAP_CS_o <= '1';

        count_readback <= READBACK_LENGTH;
        SM_RB_STATUS_o <= '0';
        reading_sm_data <= '0';
        reading_sm_data_d <= '0';

        s_selectmap_data_d <= x"FF";

        strt_rb <= '0';

        T_SELECTMAP_DATA_o <= x"FF";

    elsif(CLOCK_i'event and CLOCK_i = '1') then
```

```
T_SELECTMAP_INIT_o <= '1';
T_SELECTMAP_CS_o <= '1';
T_SELECTMAP_WRITE_o <= '1';

reading_sm_data <= '0';
reading_sm_data_d <= reading_sm_data;

-- Give it some time
if (count_readback < READBACK_DELAY) then
    count_readback <= count_readback + 1;

-- Write the readback commands to X2s selectmap interface
elsif (count_readback < READBACK_COMMAND_LENGTH +
                        READBACK_DELAY ) then
    T_SELECTMAP_CS_o <= '0';
    T_SELECTMAP_WRITE_o <= '0';
    T_SELECTMAP_DATA_o <=
            readback_command(count_readback -
    READBACK_DELAY );
    count_readback <= count_readback + 1;

--INITialize readback (deassert CS and WRITE)
elsif (count_readback < READBACK_COMMAND_LENGTH +
    READBACK_DELAY +
                        READBACK_INIT1_DELAY ) then
    T_SELECTMAP_CS_o <= '1';
    T_SELECTMAP_WRITE_o <= '1';
    count_readback <= count_readback + 1;

-- INITialize readback (assert CS)
elsif (count_readback < READBACK_OFFSET ) then
    T_SELECTMAP_CS_o <= '0';
    T_SELECTMAP_WRITE_o <= '1';
    count_readback <= count_readback + 1;

-- read back the configuration data from X2
elsif ((count_readback < (READBACK_LENGTH)) ) then

    T_SELECTMAP_CS_o <= '0';
    T_SELECTMAP_WRITE_o <= '1';
    reading_sm_data <= '1';
    -- data_d will be FF until second clock in this
    statement
    -- this prevents SM readback from hanging on the next
    statement
    -- if there happen to be no FFs at the beginning
    s_selectmap_data_d <= s_selectmap_data_i;

    -- Skip all dummy FF data coming out of the selectmap
    interface
    -- at the beginning
    if (s_selectmap_data_i /= x"FF" and
        s_selectmap_data_d = x"FF") then
        strt_rb <= '1';
    end if;
```

```
                -- Don't increment the readback counter unless we've
                cleared out
                -- all the dummy FF's (strt_rb = '1') and we we are
                latching
                -- the selectmap data in the read_sm_st of the state
                machine
                -- CCLK is clocked off of read_sm_st as well, so every
                time
                -- we are in read_sm_st, another byte of selectmap data
                is
                -- clocked out, and we grab it the next time we're in
                read_sm_st
                -- we only want to increment the counter when we
                actually
                -- grab a selectmap readback byte
                if (report_error_state = read_sm_st and strt_rb = '1' )
                then
                    count_readback <= count_readback + 1;
                end if;

            -- sit here until we get a readback request from top level
            elsif (count_readback = READBACK_LENGTH ) then
                count_readback <= READBACK_LENGTH;
                SM_RB_STATUS_o <= '0';
                strt_rb <= '0';
                -- Make sure this starts out as FF so we will start the
                readback
                -- even if there are no dummy FF's at the beginning of
                the
                -- readback
                s_selectmap_data_d <= x"FF";

                if (SM_RB_RQST_i = '1') then
                   count_readback <= 0;
                   SM_RB_STATUS_o <= '1';
                end if;

            end if;
        end if;
    end process;

end rtl;
```

G. X1 PC/104 INTERFACE VHDL PACKAGE

(from [17])

```
----------------------------------------------------------------------
-- pc104Int.vhd                                                     --
-- Author: Mindy Surratt, 2005                                      --
-- Research Associate, Naval Postgraduate School, Monterey, CA      --
--                                                                  --
-- PC/104 Interface code package                                    --
-- constants: bus addresses for the PC/104 interface                --
-- everything else unused                                           --
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

package pc104IntPack is

  --------------------------------------------------------------------
  --  PC104 Bus Address
  --------------------------------------------------------------------
CONSTANT C_IMULowAddress  : std_logic_vector(9 downto 0) :=
                "1101000000"; --x340
CONSTANT C_IMUHighAddress : std_logic_vector(9 downto 0) :=
                "1101101000"; --x368
CONSTANT C_LowAddress     : std_logic_vector(9 downto 0) :=
                "1101000000"; --x340
CONSTANT C_HighAddress    : std_logic_vector(9 downto 0) :=
                "1101101000"; --x368
CONSTANT C_RESET          : std_logic_vector(9 downto 0) :=
                "1101000010"; --x342
CONSTANT C_TEST_ADDR      : std_logic_vector(9 downto 0) :=
                "1101000001"; --x341
CONSTANT C_pix_ADDR       : std_logic_vector(9 downto 0) :=
                "1101000000"; --x340
CONSTANT C_STATUS_ADDR    : std_logic_vector(9 downto 0) :=
                "1101000011"; --x343

function TMR( a:std_logic_vector;b:std_logic_vector;c:std_logic_vector)
return std_logic_vector;

function  TMR( a:std_logic;b:std_logic;c:std_logic)
return std_logic;
end pc104IntPack;

package body pc104IntPack is

function TMR( a:std_logic_vector;b:std_logic_vector;c:std_logic_vector)
    return std_logic_vector is
    variable tmr_ret :std_logic_vector(a'range);
begin
    for i in a'range loop
        tmr_ret(i) := (a(i) and c(i)) or (a(i) and b(i)) or (b(i) and
                c(i));
    end loop;
    return tmr_ret;
```

```
end TMR;

function TMR( a:std_logic;b:std_logic;c:std_logic)
    return std_logic is
    variable tmr_ret :std_logic;
begin
    tmr_ret:= (a and c) or (a and b) or (b and  c);
    return tmr_ret;
```

H. X1 PC/104 INTERFACE CODE

(from [17])

```
-----------------------------------------------------------------------
-- pc104Int.vhd                                                      --
-- Author: Mindy Surratt, 2005                                       --
-- Research Associate, Naval Postgraduate School, Monterey, CA       --
--                                                                   --
-- PC/104 Interface code                                             --
-----------------------------------------------------------------------

library IEEE;
--library exemplar;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
--use exemplar.exemplar_1164.all;
use work.pc104intpack.all;

--
--  PC104 interface
--
entity pc104Int is
    port (
    --
    -- Inputs
    --
    Clk_i           : in std_logic;
    reset           : in std_logic;
    T_Address_i_    : in STD_LOGIC_VECTOR (9 downto 0);
    T_IORead_i      : in STD_LOGIC; --active low, read data into
                T_Data_io
    T_IOWrite_i     : in STD_LOGIC; -- active high
    T_AEN_i         : in STD_LOGIC; -- active low

    -- IO
    T_Data_io_    : inout STD_LOGIC_VECTOR (7 downto 0);

--
--  FPGA side interface
--
    pix_data_out    : out std_logic_vector(7 downto 0);
    pix_data_rdy    : out std_logic;
    pix_data_ack    : in  std_logic;

    pix_data_in     : in std_logic_vector(7 downto 0);
    pix_data_wr     : in std_logic --active high

    );
end pc104Int;

architecture rtl of pc104Int is

component fifo_out
    port (
    din: IN std_logic_VECTOR(31 downto 0);
    wr_en: IN std_logic;
    wr_clk: IN std_logic;
    rd_en: IN std_logic;
    rd_clk: IN std_logic;
    ainit: IN std_logic;
    dout: OUT std_logic_VECTOR(31 downto 0);
    full: OUT std_logic;
    empty: OUT std_logic);
end component;
--
--  SIGNALS
--
SIGNAL S_IOTemp___: STD_LOGIC_VECTOR(7 downto 0);
SIGNAL TestData __: STD_LOGIC_VECTOR(7 downto 0);
SIGNAL TestData2__: STD_LOGIC_VECTOR(7 downto 0);
SIGNAL S_Decode___: STD_LOGIC;
SIGNAL S_RangeCheck     _: STD_LOGIC;
signal iow___: std_logic;
signal iow_d___: std_logic;
signal data_in___: std_logic_vector(7 downto 0);
signal addr___: std_logic_vector(9 downto 0);
signal aen___: std_logic;
signal pix_data_in_reg__: std_logic_vector(7 downto 0);
signal status_reg__: std_logic_vector(7 downto 0);
signal status_reg_iord__: std_logic_vector(7 downto 0);
signal status_read__: std_logic;
signal status_read_d__: std_logic;
signal status_reg_d__: std_logic;
signal status_read_dd__: std_logic;
signal ioread_d___: std_logic;
signal ioread_dd___: std_logic;
signal fifo_read: std_logic;
signal fifo_read_d: std_logic;
signal fifo_read_dd: std_logic;
signal fifo_rd: std_logic;
signal full:std_logic;
signal empty:std_logic;
signal ainit:std_logic;
signal fifo_din: std_logic_vector(31 downto 0);
signal fifo_dout: std_logic_vector(31 downto 0);

begin
ainit <= '0';
--
-- sync
--
process(clk_i)
begin
if(clk_i'event and clk_i = '1' ) then
   data_in <= T_data_io;
   iow <= T_IOWrite_i;
   iow_d <= iow;
   aen<= t_aen_i;
   addr<= t_address_i;
end if;
end process;
```

```
    --
    -- Address Range Check lines
    --
    S_RangeCheck <= '1' when (  (T_Address_i >= C_LowAddress)
                            and (T_Address_i <= C_HighAddress) )
                        else '0';

    --
    -- Decode Line
    --
    S_Decode <= '1' when (T_AEN_i = '0' and S_RangeCheck = '1') else
                '0';
    --
    -- Bidirctional Data IO
    --
    T_Data_io <= S_IOTemp when (S_Decode = '1' and T_IORead_i = '0')
                          else "ZZZZZZZZ";


IOWrite: Process(clk_i)
begin
    if(clk_i'event and clk_i = '1' ) then

        if(pix_data_ack = '1') then
            pix_data_rdy <= '0';
        end if;

        if(iow = '1' and iow_d = '0') then

            if AEN = '0' then

                case Addr is

                    when C_TEST_ADDR =>
                            TestData(7 downto 0) <= data_in;

                    when C_pix_ADDR =>
                            pix_data_out(7 downto 0) <= data_in;
                            pix_data_rdy <= '1';

                    when others =>
                            NULL;

                end case;

            end if;

        end if;

    end if;
end process;

IORead: Process(T_Address_i,TestData,pix_data_in_reg,status_reg_iord)
begin

    fifo_read <= '0';
    status_read <= '0';
    case T_Address_i is

        when C_pix_ADDR=>
                S_IOTemp <=  pix_data_in_reg;
                fifo_read <= '1';

        when C_STATUS_ADDR =>
                S_IOTemp <= status_reg_iord;
                status_read <= '1';

        when others =>
                S_IOTemp <=   (others => '0');

        end case;

end process;

process(clk_i,reset)
begin

    if (reset = '1') then

        status_reg_iord(0) <= '0';

    elsif(clk_i'event and clk_i = '1') then

        fifo_read_d <= fifo_read;
        fifo_read_dd <= fifo_read_d;
        status_reg_iord(7 downto 1) <= (others => '0');

        if (fifo_rd = '1') then
            status_reg_iord(0) <= '1';
        end if;

        if (ioread_d = '1' and ioread_dd = '0' and fifo_read_dd = '1' )
                then
            status_reg_iord(0) <= '0';
        end if;

    end if;

end process;

fifo_rd <= '1' when empty = '0' and status_reg_iord(0) = '0' else '0';

process(clk_i)
begin

    if(clk_i'event and clk_i = '1') then

        status_reg(7 downto 1) <= (others => '0');
        ioread_d <= T_IORead_i;
        ioread_dd <= ioread_d;
        status_read_d <=  status_read;
        status_read_dd <=  status_read_d;
        if(pix_data_wr = '1' ) then
            status_reg(0)<= '1';
        end if;

        if (ioread_d = '1' and ioread_dd = '0' and status_reg_iord(0) =
                '1'
                and status_read = '1'  ) then
            status_reg(0) <= '0';
        end if;

    end if;

end process;

fifo_din(31 downto 8) <= (others => '0');
fifo_din(7 downto 0) <= pix_data_in;
pix_data_in_reg <= fifo_dout(7 downto 0);

fifo_out_1 : fifo_out
port map (
    din => fifo_din,
    wr_en => pix_data_wr,
    wr_clk => clk_i,
    rd_en => fifo_rd,
    rd_clk => clk_i,
    ainit => ainit,
    dout => fifo_dout,
    full => full,
    empty => empty);

end rtl;
```

I. X2 INTERFACE CODE FOR X1

(after [17])

```
-------------------------------------------------
-- filename: x2Int.vhd
-- author: Mindy Surratt (2005)
-- modified: James Coudeyras (2005)
--
-- This file is the X1 control interface to X2.
-- Any additional programs necessary to run the X2 experiment
-- are incorporated into this module.
-- This module also contains the reporting and readback
-- requirements as well as the reconfiguration threshold
--
-------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity x2Int is

-- FOR SR IMPLEMENTATION {START}
    generic (NUM_OF_TAPS     : integer := 2; -- Number of taps LFSR
        WIDTH                : integer := 15; -- Width of m-length LFSR
        NUM_SRS              : integer := 16); -- # of shift registers
                in X2
-- FOR SR IMPLEMENTATION {END}

    port (
        CLOCK_i              : in std_logic; -- 25 MHz system clock
        CLOCK_X2_i           : in std_logic; -- 12.5 MHz system clock w/
                reset
                                          -- add appropriate clock to
                clockGen
        RESET_i              : in std_logic; -- Reset signal

-- FOR SR IMPLEMENTATION {START}, signals coming directly to/from X2
-- JCC 20051031 SR copy removed from X1 on this and future versions,
                fcn implemented on X2
        T_BITOUT_TO_X2_o         : out std_logic;
        T_RESET_TO_X2_o          : out std_logic;
        T_CE_TO_X2_o             : out std_logic; -- clock enable to X2
        T_DOUT_FROM_X2_i         : in std_logic; -- verify LFSR
                operation
        T_XOR_DOUT_FROM_X2_i     : in std_logic; -- function moved to X2
        T_SUMDIFF_FROM_X2_i      : in std_logic_vector (15 downto 0);
-- FOR SR IMPLEMENTATION {END}

        DATA_o               : out std_logic_vector(7 downto 0);
          --Data bus out of X2 interface, in this case used to write to
                PC104
        DATA_i               : in std_logic_vector(7 downto 0);
          --Data bus into X2 interface, in this case used to read from
                PC104
        PC104_WR_EN_o       : out std_logic;
          -- Active high, if WR_RDY = '1', then set WR_EN = '1' for one
                clock
          --  and whatever is on DATA_o when WR_EN is high will get
                written to PC104

        PC104_WR_RDY_i      : in std_logic; --Active High, ok to write
                to PC104
          -- if WR_RDY is high, whatever you write to PC104
          -- will definitely get printed (your component has priority)

        PC104_RD_RDY_i      : in std_logic; --Active high, if RD_RDY =
                '1', then there is
          -- data on the PC104 bus ready to be read. Once you read the
                data (from DATA_i),
          -- set RD_ACK high for one clock to release the PC104

        PC104_RD_ACK_o      : out std_logic; --Active high

        SM_CONFIG_RQST_o    : out std_logic; --Active high, set
                config_rqst high
          -- for one clock if you want to start a selectmap
                config/reconfig

        SM_CONFIG_STATUS_i  : in std_logic; --Active high, stays '1' as
                long as
          -- a selectmap config is going on (don't request a readback
                or reconfig while
          -- either is still active, it won't hurt anything, but it
                won't go through)

        SM_RB_RQST_o        : out std_logic;--Active high, set rb_rqst
                high
          -- for one clock if you want to start a selectmap readback

        SM_RB_STATUS_i      : in std_logic --Active high, stays '1' as
                long as
          -- a selectmap rb is going on (don't request a readback or
                reconfig while
          -- either is still active, it won't hurt anything, but it
                won't go through)
    );
end x2Int;

architecture rtl of x2Int is

-- DLY_TIME counter and reset signals

    CONSTANT DLY_TIME           : integer := 750000000; --SIMULATION
                5000;
    -- ~30 seconds
    signal s_reset_sr           : std_logic;
    signal first_reset          : std_logic;
    signal sm_rb_status_d       : std_logic;
    signal dly_cnt              : integer range 0 to 1000000000;
    signal dly_start_rb         : std_logic;
    signal sm_config_status_d   : std_logic;

-- Error report timer signals
    CONSTANT ERR_RPT_TIME       : integer := 75000000; --heartbeat
                reports every 3 sec
      -- time between automatic error reports, adjust as needed (use 50
                for simulations)
    CONSTANT REPORT_OUT_LENGTH  : integer := 23; -- set to the number
                of bytes
      -- included in your output vector that needs to be printed to
                PC104
    signal sr_timer         : integer range 0 to ERR_RPT_TIME; -- for
                heartbeat reports
    signal count_out_vect   : integer range 0 to REPORT_OUT_LENGTH;
    signal report_out_vect  : std_logic;
-- ensure these split lines don't cause a problem
    type output_vector is array (REPORT_OUT_LENGTH-1 downto 0)
        of std_logic_vector(7 downto 0);
    signal out_vect         : output_vector;

-- readback/reconfig process, CLOCK_X2_i signals
    signal reconfig_from_error      : std_logic;
    signal sr_start_rb              : std_logic;

--readback/reconfig process, T_CLOCK_i signals
    signal reconfig_from_error_t_clock  : std_logic;
    signal reconfig_from_error_save     : std_logic;
    signal rb_started                   : std_logic;
    signal reconfig_timer               : integer range 0 to
                ERR_RPT_TIME;

-- X1 SR signals (LFSR) {START}
    signal bit_out              : std_logic;
    signal lfsr                 : std_logic_vector(WIDTH-1 downto 0) :=
                "111111111111111";
    signal par_fdbk             : std_logic_vector(NUM_OF_TAPS downto
                0); --parity fdbk
    signal lfsr_in              : std_logic; --mux

    type column is array ((NUM_SRS-1) downto 0) of std_logic_vector(7
                downto 0);

    type tap_point_array is array (NUM_OF_TAPS-1 downto 0) of integer;
    -- Parameterize LFSR taps. (e.g. g(D) = D^15 + D + 1)
    -- Tap points, including output tap 0.
    constant TAP                : tap_point_array := (1, 0);

-- X1 SR signals (SR)
    signal ce                   : std_logic := '1';
-- JCC 20051031 split ce into 30 sec smrb and reconfig from error
    signal ce_30                : std_logic := '1';
    signal ce_recon             : std_logic := '1';
    signal s_dout_from_X2_i     : std_logic_vector(7 downto 0);

-- X1 SR signals (Counter)
    signal cnt_dout_err     : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V0       : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V1        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V2        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V3        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V4        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V5        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V6        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V7        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V8        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V9        : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V10       : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V11       : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V12       : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V13       : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V14       : std_logic_vector (7 downto 0) :=x"00";
    signal cnt_col_V15       : std_logic_vector (7 downto 0) :=x"00";
-- X1 SR signals {END}

begin

-- Asynchronous assignments of top level signals
    T_RESET_TO_X2_o <= s_reset_sr;
--MLS might help if we actually use sr_start_rb...
--MLS 20051018 get rid of reconfig from error for now
    SM_RB_RQST_o <= sr_start_rb or  dly_start_rb;
--    SM_RB_RQST_o <= dly_start_rb;
    SM_CONFIG_RQST_o <= reconfig_from_error_t_clock;
--    SM_CONFIG_RQST_o <= '0';

-- Processes for SR implementation (LFSR portion) {START}
    T_BITOUT_TO_X2_o <= bit_out;

--MLS SR stop SR during readback
    T_CE_TO_X2_o <= ce; -- clock enable

-- JCC 20051031 make ce function of 30 second rb ce and reconfig from
                error ce
    ce <= (ce_30 and ce_recon);
    ce_30 <= (not SM_RB_STATUS_i); -- clock disable for 30 sec readback

    lfsr0 : process(CLOCK_X2_i) begin
        if (s_reset_sr = '1') then
            lfsr <= (others=>'1');
        elsif (CLOCK_X2_i'event and CLOCK_X2_i='1') then
            if (ce = '1') then
                lfsr <= lfsr_in & lfsr(lfsr'high downto 1);
            end if;
        end if;
    end process;

    par_fdbk(0) <= '0';

    fdbk : for X in 0 to TAP'high generate -- parity generator
        par_fdbk(X+1) <= par_fdbk(X) xor lfsr(TAP(X));
    end generate fdbk;

    lfsr_in <= par_fdbk(par_fdbk'high);
    bit_out <= lfsr(lfsr'low); -- LFSR output

-- Processes for SR implementation (LFSR portion) {END}

-- Processes for SR implementation (Counter portion) {START}

-- This process counts errors from X2, compares the output from the a-
             side of X2 SRs
-- with the a-side of X1 (counts errors), and resets error counters
    counter : process (CLOCK_X2_i, s_reset_sr)
        variable cnt_dout_var : integer range 0 to 255 := 0;

    begin

        if (s_reset_sr = '1') then
            cnt_dout_var := 0;
            cnt_dout_err <= X"00";
            cnt_col_V0 <= x"00";
            cnt_col_V1 <= x"00";
            cnt_col_V2 <= x"00";
            cnt_col_V3 <= x"00";
            cnt_col_V4 <= x"00";
            cnt_col_V5 <= x"00";
            cnt_col_V6 <= x"00";
            cnt_col_V7 <= x"00";
            cnt_col_V8 <= x"00";
            cnt_col_V9 <= x"00";
            cnt_col_V10 <= x"00";
            cnt_col_V11 <= x"00";
            cnt_col_V12 <= x"00";
            cnt_col_V13 <= x"00";
            cnt_col_V14 <= x"00";
            cnt_col_V15 <= x"00";

        elsif (CLOCK_X2_i'event and CLOCK_X2_i='1') then

--MLS SR don't do this if we're reading back, will get nonsense
            if (ce = '1') then

                s_dout_from_X2_i <= "0000000" & T_DOUT_FROM_X2_i;

                if (T_XOR_DOUT_FROM_X2_i = '1') then
                    cnt_dout_var := cnt_dout_var + 1;
                end if;
                cnt_dout_err <=
                std_logic_vector(to_unsigned(cnt_dout_var,8));

                if T_SUMDIFF_FROM_X2_i(0)='1' then
                    cnt_col_V0 <= cnt_col_V0 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(1)='1' then
                    cnt_col_V1 <= cnt_col_V1 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(2)='1' then
                    cnt_col_V2 <= cnt_col_V2 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(3)='1' then
                    cnt_col_V3 <= cnt_col_V3 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(4)='1' then
                    cnt_col_V4 <= cnt_col_V4 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(5)='1' then
                    cnt_col_V5 <= cnt_col_V5 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(6)='1' then
                    cnt_col_V6 <= cnt_col_V6 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(7)='1' then
                    cnt_col_V7 <= cnt_col_V7 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(8)='1' then
                    cnt_col_V8 <= cnt_col_V8 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(9)='1' then
                    cnt_col_V9 <= cnt_col_V9 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(10)='1' then
                    cnt_col_V10 <= cnt_col_V10 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(11)='1' then
                    cnt_col_V11 <= cnt_col_V11 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(12)='1' then
                    cnt_col_V12 <= cnt_col_V12 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(13)='1' then
                    cnt_col_V13 <= cnt_col_V13 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(14)='1' then
                    cnt_col_V14 <= cnt_col_V14 + 1;
                end if;
                if T_SUMDIFF_FROM_X2_i(15)='1' then
                    cnt_col_V15 <= cnt_col_V15 + 1;
                end if;
            end if;
        end if;
    end process;
-- Processes for SR implementation (Counter portion) {END}

-- Timer to determine how frequently to print out heart beat error
                reports (every 3 sec)
    process(CLOCK_i, s_reset_sr) begin
        if (s_reset_sr = '1') then
            sr_timer <= 0;
        elsif(CLOCK_i'event and CLOCK_i = '1') then
            if (sr_timer = ERR_RPT_TIME) then
                sr_timer <= 0;
            else
                sr_timer <= sr_timer + 1;
            end if;
        end if;
    end process;
```

```
-- Do reset for sr
    process (CLOCK_i,RESET_i) begin
        if (RESET_i = '1') then
            s_reset_sr <= '1';
        elsif (CLOCK_i'event and CLOCK_i = '1') then
            sm_config_status_d <= SM_CONFIG_STATUS_i;

--MLS hold reset high for 30 seconds, do reset after config
--MLS 2005.09.15 only hold reset high until version is done and x2 is
                done configuring, no need
-- to hold everything off for 30s
            s_reset_sr <= '0';
--MLS 2005.10.18 reset counters and sr as soon as a reconfig happens
            if (first_reset='0' or (SM_CONFIG_STATUS_i = '0' and
                sm_config_status_d = '1' ) ) then
                s_reset_sr <= '1';
            end if;
        end if;
    end process;

--MLS 2005.09.15 wait until after X2 is done configuring to do first
                reset (instead of 30s)
    process(CLOCK_i,RESET_i) begin
        if (RESET_i = '1') then
            first_reset <= '0';
        elsif (CLOCK_i'event and CLOCK_i = '1') then
            if (first_reset = '0' and SM_CONFIG_STATUS_i = '0' and
                sm_config_status_d = '1') then
                first_reset <= '1';
            end if;
        end if;
    end process;

-- Every DLY_TIME clocks after that, do a selectmap readback
-- don't start dly_cnt until s_reset_sr goes low
-- (after version is done and x2 config is done)
    process(CLOCK_i, s_reset_sr)
    begin
        if (s_reset_sr = '1') then
            sm_rb_status_d <= '0';
            dly_cnt <= 0;
            dly_start_rb <= '0'; -- Signal to notify that the delay
                counter wants
                                 -- to perform a SMRB

        elsif(CLOCK_i'event and CLOCK_i = '1') then

            dly_start_rb <= '0';
            sm_rb_status_d <= SM_RB_STATUS_i;

-- If we just finished a SMRB, start the counter over
            if (SM_RB_STATUS_i = '0' and sm_rb_status_d = '1') then
               dly_cnt <= 0;
            elsif (dly_cnt < DLY_TIME) then
                dly_cnt <= dly_cnt + 1;
            elsif (dly_cnt = DLY_TIME) then
                dly_cnt <= 0;
                dly_start_rb <= '1'; -- After DLY_TIME clocks, start a
                readback
            end if;
        end if;
    end process;
-- SMRB strictly based on time (not certain number of errors

--Process to write error reports out to the PC104
    process (CLOCK_i, s_reset_sr) begin
        if (s_reset_sr = '1') then --hold off printing error reports
                until SR is reset
            count_out_vect <= 0;
            report_out_vect <= '0';
            PC104_WR_EN_o <= '0';
            DATA_o <= x"31";
        elsif (CLOCK_i'event and CLOCK_i = '1') then

            PC104_WR_EN_o <= '0'; --default assignment for WR_EN

-- Whenever we've gone through ERR_RPT_TIME clocks or we get an error
                from X2 (a signal
-- coming directly from X2), and we've already finished printing out
                the last error report
-- (report_out_vect = '0'), set the output error report vector to the
                correct values
            if ( (reconfig_from_error = '1' and report_out_vect = '0')
                or ( sr_timer = ERR_RPT_TIME and report_out_vect = '0'
                and
                    reconfig_from_error_save = '0' and
                SM_CONFIG_STATUS_i = '0' ) ) then
                        -- send out_vect to PC104
                out_vect(0) <= x"45"; --E
                out_vect(1) <= x"52"; --R, easier to spot error reports
                in output
                out_vect(2) <= s_dout_from_X2_i(7 downto 0);
                out_vect(3) <= s_dout_from_X2_i(7 downto 0); -- extra
                output vector
                out_vect(4) <= T_SUMDIFF_FROM_X2_i(15 downto 8);
                out_vect(5) <= T_SUMDIFF_FROM_X2_i(7 downto 0);
                out_vect(6) <= cnt_dout_err;
                out_vect(7) <= cnt_col_V15;
                out_vect(8) <= cnt_col_V14;
                out_vect(9) <= cnt_col_V13;
                out_vect(10) <= cnt_col_V12;
                out_vect(11) <= cnt_col_V11;
                out_vect(12) <= cnt_col_V10;
                out_vect(13) <= cnt_col_V9;
                out_vect(14) <= cnt_col_V8;
                out_vect(15) <= cnt_col_V7;
                out_vect(16) <= cnt_col_V6;
                out_vect(17) <= cnt_col_V5;
                out_vect(18) <= cnt_col_V4;
                out_vect(19) <= cnt_col_V3;
                out_vect(20) <= cnt_col_V2;
                out_vect(21) <= cnt_col_V1;
                out_vect(22) <= cnt_col_V0;
                report_out_vect <= '1';
            end if;

-- If we've set the output vector (report_out_vect = '1'), then print
                the output vector
-- to the PC104 one byte at a time (REPORT_OUT_LENGTH bytes will be
                printed)
-- Be sure to set REPORT_OUT_LENGTH to proper value in signal
                definitions above
            if (report_out_vect='1') then
                if (count_out_vect < REPORT_OUT_LENGTH and
                PC104_WR_RDY_i = '1') then
--MLS
--                    ce <= '0'; -- disables shift registers while
                reading errors
                    DATA_o <= out_vect(count_out_vect);
                    PC104_WR_EN_o <= '1';
                    count_out_vect <= count_out_vect + 1;
                elsif (count_out_vect = REPORT_OUT_LENGTH) then
--MLS
--                    ce <= '1'; -- enables shift registers after
                report finished
                    count_out_vect <= 0;
                    report_out_vect <= '0';
                end if;
            end if;
        end if;
    end process;

    process(CLOCK_X2_i,s_reset_sr) begin

        if (s_reset_sr = '1') then

            reconfig_from_error <= '0';

        elsif (CLOCK_X2_i'event and CLOCK_X2_i = '1') then

            reconfig_from_error <= '0';

            if ( (cnt_col_V0 = x"80" or cnt_col_V1 = x"80" or
                cnt_col_V2 = x"80" or
                    cnt_col_V3 = x"80" or  cnt_col_V4 = x"80" or
                cnt_col_V5 = x"80" or
                    cnt_col_V6 = x"80" or  cnt_col_V7 = x"80" or
                cnt_col_V8 = x"80" or
                    cnt_col_V9 = x"80" or  cnt_col_V10 = x"80" or
                cnt_col_V11 = x"80" or
                    cnt_col_V12 = x"80" or  cnt_col_V13 = x"80" or
                cnt_col_V14 = x"80" or
                    cnt_col_V15 = x"80" or cnt_dout_err = x"80") and
                reconfig_from_error = '0' and reconfig_from_error_save =
                '0' ) then
                reconfig_from_error <= '1';
            end if;

        end if;

    end process;
```

```
--process to trigger SMRB/RC on errors
    process(CLOCK_i, s_reset_sr) begin
        if (s_reset_sr = '1') then

            reconfig_timer <= 0;
            reconfig_from_error_save <= '0';
            reconfig_from_error_t_clock <= '0';
            sr_start_rb <= '0'; --signal to notify the sr counter has
                reported
                                -- enough errors to warrant a readback
            rb_started <= '0'; --make sure sr_start_rb only 1 clock
            ce_recon <= '1';

        elsif (CLOCK_i'event and CLOCK_i = '1') then

            sr_start_rb <= '0';
            reconfig_timer <= reconfig_timer + 1;
            reconfig_from_error_t_clock <= '0';

-- If we got a reconfig request from the sr, hold that thought
            if (reconfig_from_error = '1') then
                reconfig_timer <= 0;
                reconfig_from_error_save <= '1';
                ce_recon <= '0';

-- Wait until SMRB is done, and then request a reconfig from the top
                level
            elsif ( reconfig_from_error_save = '1' and SM_RB_STATUS_i =
                '0' and
                    sm_rb_status_d = '1' ) then
                reconfig_from_error_t_clock <= '1';
                reconfig_from_error_save <= '0';
                rb_started <= '0';

-- Wait until last error report is printed out before starting readback
-- Once readback has started, don't start another one!
-- Wait to start readback for ~3s (50000000) after reaching critical
-- number of errors, this is for JTAG external error injection, errors
-- begin accumulating before it is done programming, so 128 errors
-- could be reached before partial reconfig complete, so tries to
-- readback while JTAG still going on.
            elsif (reconfig_from_error_save = '1' and reconfig_timer =
                ERR_RPT_TIME and
                    report_out_vect = '0' and SM_RB_STATUS_i = '0' and
                rb_started = '0' ) then
                sr_start_rb <= '1';
                rb_started <= '1';

            end if;

        end if;
    end process;

end rtl;
```

J. CLOCK CODE

(after [17])

```
-------------------------------------------------
-- filename: clockGen.vhd
-- author: Mindy Surratt (2005)
-- modified: James Coudeyras (2005)
--
-- This file divides the 25 MHz clock down to 12.5 MHz
--
-------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity clockGen is

    port (

        T_CLOCK_i           : in std_logic;

        CLOCK_o             : out std_logic;
        CLOCK_X2_o          : out std_logic;
        RESET_FROM_X2INT_i  : in std_logic;

        CLOCK_NOBUFG_o      : out std_logic
    );

end clockGen;

architecture rtl of clockGen is

--REMOVE FOR SIMULATION
    component BUFG port (
        I                   : in std_logic;
        O                   : out std_logic);
    end component;

    signal s_clock              : std_logic;
    signal s_clock_i            : std_logic := '0';
    signal s_clock_X2           : std_logic;
    signal s_clock_X2_i         : std_logic := '0';
    signal s_clock_X2_cnt       : integer range 0 to 16;

begin

    CLOCK_NOBUFG_o <= s_clock_i;

    process(t_clock_i)
    begin
        if(T_CLOCK_i'event and T_CLOCK_i = '1') then
            s_clock_i <= not s_clock_i;
        end if;
    end process;

-- FOR SR IMPLEMENTATION {START}
-- (used because this clock has a reset)
    process (T_CLOCK_i,RESET_FROM_X2INT_i) begin
        if (RESET_FROM_X2INT_i = '1') then
            s_clock_x2_i <= '0'; -- changed _cnt to _i
        elsif (T_CLOCK_i'event and T_CLOCK_i = '1') then
            if (s_clock_X2_cnt >= 1) then  -- this achieves a divide by
                2 clock period!!!
                s_clock_X2_i <= not s_clock_X2_i;
                s_clock_X2_cnt <= 0;
            else
                s_clock_X2_cnt <= s_clock_X2_cnt + 1;
            end if;
        end if;
    end process;
-- FOR SR implementation {END}

--MLS REMOVE FOR SIMULATION
--s_clock_X2 <= s_clock_X2_i;
--s_clock <= s_clock_i;
--Clock distribution network for X2 clock
    s_clock_X2_bufg : BUFG port map ( -- comment out this code only for
                simulations!!!
        I           => s_clock_X2_i,
        O           => CLOCK_X2_o
    );
--Clock distribution network for 25MHz system clock
    s_clock_bufg : BUFG port map ( -- comment out this code only for
                simulations!!!
        I           => s_clock_i,
        O           => CLOCK_o
    );

end rtl;
```

K. X1 TOP-LEVEL CODE

(after [17])

```
-------------------------------------------------
-- filename: cftp_x1.vhd
-- author: Mindy Surratt (2005)
-- modified: James Coudeyras (2005)
--
-- This is the top level code for X1
--
-------------------------------------------------

-- Modifications for different X2 implementations:
-- only have to change signal names/sizes coming from X2
-- must be changed in top level entity, x2Int component
-- declaration, and x2Int port map.  search for "implementation"
-- each instance that needs to be changed is surrounded
-- by comments:
-- for XXXXX implementation
-- Also change signal names in ucf file


--Naming conventions:
-- Top level signals that go directly to pins, or pass directly through
--  the top level to/from a component to pins
--          T_XXXXXX_io T_XXXXXX_i  T_XXXXX_o
-- Top level signals that do not pass directly to pins (ie, internal
--  signals between components/top level
--          XXXXXXX_io  XXXXX_i   XXXXXX_o
-- Constants
--          XXXXXXXX
-- Internal signals
--          xxxxxxxx
-- internal signal that is just a copy or slight modification
--   of top level signal
--          s_xxxxx_io  s_xxxxx_i  s_xxxxx_o
-- delayed copy of signal (ie signal_d <= signal)
--          xxxxx_d
-- state machine signal
--          xxxxx_state
-- state machine state
--          xxxxx_st
-- counter
--          cnt_xxxxx
-- start signal
--          start_xxxxx
-- done signal
--          done_xxxxx


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
```

```
entity cftp_x1 is
    port (

-- To/From X2 for SR implementation {START}
-- signals going to pins on X2
-- change/add/remove as needed
-- also change cftp_x1.ucf file to match
        T_BITOUT_TO_X2_o        : out std_logic;
        T_RESET_TO_X2_o         : out std_logic;
        T_CE_TO_X2_o            : out std_logic; -- clock enable to X2
        T_DOUT_FROM_X2_i        : in std_logic; --_vector (15 downto
                0);
        T_XOR_DOUT_FROM_X2_i    : in std_logic; -- comparing douts on
                X2
        T_SUMDIFF_FROM_X2_i     : in std_logic_vector (15 downto 0);
-- To/From X2 for SR implementation {END}

--PC/104 Signals
        T_clock_i           : in STD_LOGIC;
        T_Address_i         : in STD_LOGIC_VECTOR (9 downto 0);
        T_IORead_i          : in STD_LOGIC;
        T_IOWRITE_i         : in STD_LOGIC;
        T_AEN_i             : in STD_LOGIC;
        T_Data_io           : inout STD_LOGIC_VECTOR (7 downto 0);

        T_X2_MODE           : out STD_LOGIC_VECTOR(2 downto 0);

--Selectmap Signals
        T_CCLK_o            : out STD_LOGIC;
        T_SELECTMAP_INIT_o  : out STD_LOGIC;
        T_SELECTMAP_WRITE_o : out STD_LOGIC;
        T_SELECTMAP_CS_o    : out STD_LOGIC;
        T_SELECTMAP_DATA_io : inout STD_LOGIC_VECTOR(7 downto 0);
        T_SELECTMAP_BUSY_i  : in STD_LOGIC;

--Flash Interface Signals
        T_FLASH_DATA_i      : in STD_LOGIC_VECTOR (15 downto 0);
        T_FLASH_ADDRESS_o   : out STD_LOGIC_VECTOR (20 downto 0);
        T_FLASH_WE_o        : out STD_LOGIC;
        T_FLASH_RP_o        : out STD_LOGIC;
        T_FLASH_WP_o        : out STD_LOGIC;
-- JCC implemented change from Mindy that allows for second flash on
                CFTP-2
--        T_FLASH_CE_o        : out STD_LOGIC;
                  T_FLASH_CE_A_o              : out STD_LOGIC;
        -- NO FLASH B on CFTP1 !!!!
                  T_FLASH_CE_B_o              : out STD_LOGIC;
        T_FLASH_OE_o        : out STD_LOGIC
    );
end cftp_x1;

architecture rtl of cftp_x1 is

    component clockGen port (

        T_CLOCK_i           : in std_logic;
        CLOCK_o             : out std_logic;
        CLOCK_X2_o          : out std_logic;
        RESET_FROM_X2INT_i  : in std_logic;

        CLOCK_NOBUFG_o      : out std_logic );

    end component;

    component x2Int port (

        CLOCK_i             : in std_logic; --25 MHz system clock
        CLOCK_X2_i          : in std_logic;
        RESET_i             : in std_logic;

-- for SR implementation {START}
-- signals going to pins on X2
-- change/add/remove as needed
-- also change cftp_x1.ucf file to match
        T_BITOUT_TO_X2_o        : out std_logic;
        T_RESET_TO_X2_o         : out std_logic;
        T_CE_TO_X2_o            : out std_logic; -- clock enable to X2
        T_DOUT_FROM_X2_i        : in std_logic; --_vector (15 downto
                0);
        T_XOR_DOUT_FROM_X2_i    : in std_logic; -- comparing douts on
                X2
        T_SUMDIFF_FROM_X2_i     : in std_logic_vector (15 downto 0);
-- for SR implementation {END}

--MLS 20051109
        STALL               : in std_logic;
--MLS END 20051109

        DATA_o              : out std_logic_vector(7 downto 0);
        DATA_i              : in std_logic_vector(7 downto 0);
        PC104_WR_EN_o       : out std_logic;
        PC104_WR_RDY_i      : in std_logic;
        PC104_RD_RDY_i      : in std_logic;
        PC104_RD_ACK_o      : out std_logic;

        SM_CONFIG_RQST_o    : out std_logic;
        SM_CONFIG_STATUS_i  : in std_logic;
        SM_RB_RQST_o        : out std_logic;
        SM_RB_STATUS_i      : in std_logic );

    end component;

    component SELECTMAP_readback port (

        T_CLOCK_i           : in std_logic;
        CLOCK_i             : in std_logic;
        CLOCK_NOBUFG_i      : in std_logic;
        RESET_i             : in std_logic;

        T_CCLK_o            : out std_logic;
        T_SELECTMAP_INIT_o  : out std_logic;
        T_SELECTMAP_WRITE_o : out std_logic;
        T_SELECTMAP_CS_o    : out std_logic;
        T_SELECTMAP_DATA_o  : out std_logic_vector(7 downto 0);
        T_SELECTMAP_DATA_i  : in std_logic_vector(7 downto 0);

        SM_RB_RQST_i        : in std_logic;
        SM_RB_STATUS_o      : out std_logic;

        T_FLASH_DATA_i      : in std_logic_vector(15 downto 0);
        T_FLASH_ADDRESS_o   : out std_logic_vector(21 downto 0);

        PC104_WR_RDY_i      : in std_logic;
        PC104_WR_EN_o       : out std_logic;
        DATA_o              : out std_logic_vector(7 downto 0) );

    end component;

    component SELECTMAP_config port (

        T_CLOCK_i           : in std_logic;
        RESET_i             : in std_logic;

        T_SELECTMAP_INIT_o  : out std_logic;
        T_SELECTMAP_WRITE_o : out std_logic;
        T_SELECTMAP_CS_o    : out std_logic;
        T_SELECTMAP_DATA_o  : out std_logic_vector(7 downto 0);

        SM_CONFIG_RQST_i    : in std_logic;
        SM_CONFIG_STATUS_o  : out std_logic;

        T_FLASH_DATA_i      : in std_logic_vector(15 downto 0);
        T_FLASH_ADDRESS_o   : out std_logic_vector(21 downto 0) );

--MLS 20051109
        PC104_WR_RDY_i      : in std_logic;
        PC104_WR_EN_o       : out std_logic;
        DATA_o              : out std_logic_vector(7 downto 0) );
--MLS END 20051109

    end component;

    component pc104Int port (

        T_CLOCK_i           : in std_logic;
        RESET_i             : in std_logic;
        T_Address_i         : in STD_LOGIC_VECTOR (9 downto 0);
        T_IORead_i          : in STD_LOGIC;
        T_IOWRITE_i         : in STD_LOGIC;
        T_AEN_i             : in STD_LOGIC;
        T_Data_io           : inout STD_LOGIC_VECTOR (7 downto 0);

        DATA_o              : out std_logic_vector(7 downto 0);
        DATA_RDY_o          : out std_logic;
        DATA_ACK_i          : in  std_logic;

        DATA_i              : in std_logic_vector(7 downto 0);
        DATA_WREN_i         : in std_logic;
        FIFO_FULL_o         : out std_logic );
    end component;

    component version port (
        Clk                 : in          std_logic;
        rstn                : in        std_logic;
        start               : in        std_logic;
        Rdy                 : out        std_logic;
        pc104_wr_rdy        : in        std_logic;
        DOut                : out        std_logic_vector(7 downto 0);
        Done                : out        std_logic );
    end component;

-- clock/reset signals

    signal s_clock          : std_logic;
    signal s_clock_nobufg   : std_logic := '0';

    signal s_clock_X2       : std_logic;

    signal s_reset          : std_logic := '1';
    signal s_reset_x2       : std_logic := '1';
    signal s_reset_d        : std_logic := '1';
-- MLS 20051109
    signal stall                    : std_logic;
    signal stall_d                    : std_logic;
-- MLS 20051109

--Signals for X2 Interface

    signal ver_done_reset           : std_logic;
--    signal x2_reset                 : std_logic;
    signal X2_wr_rdy                : std_logic;
    signal X2_data_to_pc104         : std_logic_vector(7 downto 0);
    signal X2_data_to_pc104_wren    : std_logic;
    signal X2_data_from_pc104_ack   : std_logic;
    signal X2_sm_config_rqst        : std_logic;
    signal X2_sm_rb_rqst            : std_logic;


--Required signals for component version_0

    signal version_reset            : std_logic;
    signal version_start            : std_logic;
    signal version_rdy              : std_logic;
    signal version_wr_rdy           : std_logic;
    signal version_out              : std_logic_vector (7 downto 0);
    signal version_done             : std_logic;
    signal version_done_d           : std_logic;


-- Signals for PC/104 Interface

    signal data_from_pc104_rdy      : std_logic;
--    signal data_from_pc104_rdy_d    : std_logic; -- NOT USED
    signal data_from_pc104_ack      : std_logic;
    signal data_from_pc104          : std_logic_vector(7 downto 0);
    signal data_to_pc104            : std_logic_vector(7 downto 0);
    signal data_to_pc104_wren       : std_logic;
    signal pc104_fifo_full          : std_logic;


-- Signals for Config module

    signal selectmap_config_rqst        : STD_LOGIC;
    signal config_active                : std_logic;
--    signal config_active_d                : std_logic; -- NOT USED

    signal flash_address_config         : std_logic_vector (21 downto
            0);

    signal INIT_config                  : STD_LOGIC;
    signal CS_config                    : STD_LOGIC;
    signal WRITE_config                 : std_logic;
    signal selectmap_write_data_config  : std_logic_vector(7 downto 0);

-- MLS 20051109
    signal SC_data_to_pc104                 : std_logic_vector(7 downto
            0);
    signal SC_data_to_pc104_wren            : std_logic;
    signal SC_wr_rdy                        : std_logic;
-- MLS END 20051109

-- Signals for Readback module

    signal RB_data_to_pc104                 : std_logic_vector(7 downto
            0);
    signal RB_data_to_pc104_wren            : std_logic;
    signal RB_wr_rdy                        : std_logic;

    signal flash_address_rb                 : std_logic_vector (21
            downto 0);

    signal selectmap_readback_rqst          : STD_LOGIC;
    signal readback_active                  : std_logic;

    signal CCLK_readback                    : std_logic;
    signal INIT_readback                    : STD_LOGIC;
    signal CS_readback                      : STD_LOGIC;
    signal WRITE_readback                   : std_logic;
    signal selectmap_write_data_readback    : std_logic_vector(7 downto
            0);

-- Selectmap signals

    signal selectmap_write_data   : std_logic_vector(7 downto 0) :=
            x"00"; --SIMULATION
    signal s_selectmap_data_i     : std_logic_vector(7 downto 0);
    signal s_selectmap_WRITE_o    : std_logic := '1'; --SIMULATION


begin
```

```
    process(s_clock)
    begin
        if(s_clock'event and s_clock = '1') then
            s_reset <= '0';
            s_reset_d <= s_reset;
        end if;
    end process;

    ver_done_reset <= not version_done;

--set mode pins on X2 to activate SelectMap mode
T_x2_mode(0) <= '0';
T_x2_mode(1) <= '1';
T_x2_mode(2) <= '1';

T_RESET_TO_X2_o <= s_reset_x2;

--signal assignments to pins

-- T_RESET_TO_X2_o <= x2_reset;

T_FLASH_RP_o <= not s_reset_d;
T_FLASH_OE_o <= '0';
T_FLASH_WP_o <= '1';
T_FLASH_WE_o <= '1';

T_FLASH_ADDRESS_o <= flash_address_config(20 downto 0) when
                config_active = '1' else flash_address_rb(20 downto 0);
T_FLASH_CE_A_o <= flash_address_config(21) when config_active = '1'
    else flash_address_rb(21) when readback_active = '1'
    else '1';
T_FLASH_CE_B_o <= not flash_address_config(21) when config_active = '1'
    else not flash_address_rb(21) when readback_active = '1'
    else '1';

s_selectmap_data_i <= T_SELECTMAP_DATA_io;
T_SELECTMAP_DATA_io   <= selectmap_write_data         when
                s_selectmap_WRITE_o = '0'  else (others => 'Z');

selectmap_write_data<= selectmap_write_data_readback when
                readback_active = '1'    else
                selectmap_write_data_config;
s_selectmap_WRITE_o <= WRITE_readback               when
                readback_active = '1'     else WRITE_config;
T_SELECTMAP_INIT_o  <= INIT_readback                when
                readback_active = '1'     else INIT_config;
T_SELECTMAP_CS_o    <= CS_readback                  when
                readback_active = '1'     else CS_config;

T_SELECTMAP_WRITE_o   <= s_selectmap_WRITE_o;

T_CCLK_o <= CCLK_readback;

--MLS 20051109
data_to_pc104       <= version_out when version_done = '0' else
                RB_data_to_pc104 when readback_active = '1' else
                SC_data_to_pc104 when config_active = '1' else
                X2_data_to_pc104;
data_to_pc104_wren  <= version_rdy when version_done = '0' else
                RB_data_to_pc104_wren when readback_active = '1' else
                SC_data_to_pc104_wren when config_active = '1' else
                X2_data_to_pc104_wren;
--MLS END 20051109

--MLS 20051109
X2_wr_rdy <= (not pc104_fifo_full) and (not readback_active ) and (not
                config_active);
RB_wr_rdy <= (not pc104_fifo_full) and (not config_active);
SC_wr_rdy <= (not pc104_fifo_full) and (not readback_active) and
                (config_active);
version_wr_rdy <= (not pc104_fifo_full) and (not readback_active ) and
                (not config_active) ;
--MLS END 20051109

-- MLS 20051109
-- read commands from pc104
process(s_clock, s_reset)
begin
    if (s_reset = '1') then
        data_from_pc104_ack <= '0';
        stall <= '0';
        stall_d <= '0';
    elsif (s_clock'event and s_clock = '1') then
        data_from_pc104_ack <= '0';
        stall_d <= stall;
        if (data_from_pc104_rdy = '1') then
            data_from_pc104_ack <= '1';
            if (data_from_pc104 = x"73") then --lowercase 's'
                stall <= '1';
            elsif (data_from_pc104 = x"72") then --lowercase 'r'
                stall <= '0';
            end if;
        end if;
    end if;
end process;
-- MLS END 20051109


-- Process to choose whether we want to readback or configure But not
                both !!!
process(s_clock, s_reset)
begin
    if (s_reset = '1') then

        selectmap_readback_rqst <= '0';
        selectmap_config_rqst <= '0';

    elsif (s_clock'event and s_clock = '1') then

        selectmap_readback_rqst <= '0';
        selectmap_config_rqst <= '0';

        version_done_d <= version_done;
```

```
-- MLS 20051109
        if ( (X2_sm_rb_rqst = '1' or (stall = '0' and stall_d = '1'))
                and  version_done = '1' and config_active = '0' and
                selectmap_config_rqst  = '0' and stall = '0'  )   then
            selectmap_readback_rqst <= '1';
        elsif ( (version_done = '1' and readback_active = '0' and
                selectmap_readback_rqst = '0' and stall = '0') and
                ((version_done = '1' and version_done_d = '0') or (
                X2_sm_config_rqst = '1' )  )  ) then
            selectmap_config_rqst <= '1';
        end if;
-- MLS END 20051109

    end if;
end process;


    clockGen0: clockGen port map (

        T_CLOCK_i           => T_CLOCK_i,--50MHz oscillator

        CLOCK_o             => s_clock, --system 25MHz clock

-- for SR implementation {START}
--if you need a different clock, change clockGen.vhd
        CLOCK_X2_o          => s_clock_X2, -- 12.5 MHz clock
        RESET_FROM_X2INT_i  => s_reset_x2,
-- for SR implementation {END}

        CLOCK_NOBUFG_o      => s_clock_nobufg -- for readback, setting
                CCLK

    );

    pc104Int0: pc104Int port map (

        T_CLOCK_i       => s_clock,    --: in std_logic;
        RESET_i         => s_reset,
        T_Address_i     => T_Address_i,    --: in STD_LOGIC_VECTOR (9
                downto 0);
        T_IORead_i      => T_IORead_i,    --: in STD_LOGIC;
        T_IOWRITE_i     => T_IOWRITE_i,    --: in STD_LOGIC;
        T_AEN_i         => T_AEN_i,    --: in STD_LOGIC;
        T_Data_io       => T_Data_io,    --: inout STD_LOGIC_VECTOR (7
                downto 0)

        DATA_o          => data_from_pc104,    --: out
                std_logic_vector(7 downto 0);
        DATA_RDY_o      => data_from_pc104_rdy,    --: out std_logic;
        DATA_ACK_i      => data_from_pc104_ack,    --: in  std_logic;
        DATA_i          => data_to_pc104,    --: in std_logic_vector(7
                downto 0);
        DATA_WREN_i     => data_to_pc104_wren,--: in std_logic;
        FIFO_FULL_o     => pc104_fifo_full
    );
```

```
version_start <= '1';
version_reset <= not s_reset;

version_0: version
--Start: active high: when '1', ready to begin reading chars
--Rdy: active high: when '1', data available from version_0
--Done: active high: when '1', all chars have been read
    port map (
        Clk             => s_clock,    --    : in          std_logic;
        rstn            => version_reset,    --    : in
                std_logic;
        start           => version_start,    --    : in
                std_logic;
        Rdy             => version_rdy,    --    : out
                std_logic;
        pc104_wr_rdy    => version_wr_rdy, -- in std_logic;
        DOut            => version_out,    --    : out
                std_logic_vector(7 downto 0);
        Done            => version_done    --    : out
                std_logic;
    );

    SELECTMAP_config0: selectmap_config port map (

        T_CLOCK_i            => s_clock,         --: in std_logic;
        RESET_i              => s_reset,         --: in std_logic;

        T_SELECTMAP_INIT_o  => INIT_config,      --: out std_logic;
        T_SELECTMAP_WRITE_o => WRITE_config,     --: out std_logic;
        T_SELECTMAP_CS_o    => CS_config,        --: out std_logic;
        T_SELECTMAP_DATA_o  => selectmap_write_data_config,      --:
                out std_logic_vector(7 downto 0);

        SM_CONFIG_RQST_i    => selectmap_config_rqst, --: in std_logic;
        SM_CONFIG_STATUS_o  => config_active,       --: out std_logic;

        T_FLASH_DATA_i      => T_FLASH_DATA_i,   --: in
                std_logic_vector(15 downto 0);
        T_FLASH_ADDRESS_o   => flash_address_config --: out
                std_logic_vector(20 downto 0) );

    );

    selectmap_readback0: SELECTMAP_readback port map (

        T_CLOCK_i           => T_CLOCK_i,
        CLOCK_i             => s_clock,           --: in std_logic;
        CLOCK_NOBUFG_i      => s_clock_nobufg,
        RESET_i             => s_reset,             --: in std_logic;

        T_CCLK_o            => CCLK_readback,       --: out std_logic;
        T_SELECTMAP_INIT_o  => INIT_readback,       --: out std_logic;
        T_SELECTMAP_WRITE_o => WRITE_readback,      --: out std_logic;
        T_SELECTMAP_CS_o    => CS_readback,         --: out std_logic;
        T_SELECTMAP_DATA_o  => selectmap_write_data_readback,--: out
                std_logic_vector(7 downto 0);
        T_SELECTMAP_DATA_i  => s_selectmap_data_i,      --: in
                std_logic_vector(7 downto 0);

        SM_RB_RQST_i        => selectmap_readback_rqst, --: in
                std_logic;
        SM_RB_STATUS_o      => readback_active,     --: out std_logic;

        T_FLASH_DATA_i      => T_FLASH_DATA_i,      --: in
                std_logic_vector(15 downto 0);
        T_FLASH_ADDRESS_o   => flash_address_rb,    --: out
                std_logic_vector(20 downto 0);

        PC104_WR_RDY_i      => RB_wr_rdy,           --: in std_logic;
        PC104_WR_EN_o       => RB_data_to_pc104_wren,--: out std_logic;
        DATA_o              => RB_data_to_pc104     --: out
                std_logic_vector(7 downto 0) );

    );


    x2Int0 : x2Int port map (

        CLOCK_i             => s_clock,         --: in std_logic;
        CLOCK_X2_i          => s_clock_X2,      -- 12.5MHz clock for
                Cordic
        RESET_i             => ver_done_reset,  --: in std_logic;

-- for SR implementation
-- signals going to pins on X2
-- change/add/remove as needed
-- also change cftp_x1.ucf file to match
        T_BITOUT_TO_X2_o        => T_BITOUT_TO_X2_o,   -- : out
                std_logic;
        T_RESET_TO_X2_o         => s_reset_x2,
        T_CE_TO_X2_o            => T_CE_TO_X2_o,   -- : out std_logic;
        T_DOUT_FROM_X2_i        => T_DOUT_FROM_X2_i,   -- : in
                std_logic;
        T_XOR_DOUT_FROM_X2_i    => T_XOR_DOUT_FROM_X2_i, -- : in
                std_logic;
        T_SUMDIFF_FROM_X2_i     => T_SUMDIFF_FROM_X2_i,  -- : in
                std_logic_vector(31 downto 0);
-- for SR implementation

        DATA_o              => X2_data_to_pc104,            -- : out
                std_logic_vector(7 downto 0);
        DATA_i              => data_from_pc104,             -- : in
                std_logic_vector(7 downto 0);
        PC104_WR_EN_o       => X2_data_to_pc104_wren,       -- : out
                std_logic;
        PC104_WR_RDY_i      => X2_wr_rdy,             -- : in
                std_logic;
        PC104_RD_RDY_i      => data_from_pc104_rdy,         -- : in
                std_logic;
        PC104_RD_ACK_o      => X2_data_from_pc104_ack,      -- : out
                std_logic;

        SM_CONFIG_RQST_o    => X2_sm_config_rqst,   -- : out std_logic;
        SM_CONFIG_STATUS_i  => config_active,          -- : in
                std_logic;
        SM_RB_RQST_o        => X2_sm_rb_rqst,       -- : out std_logic;
        SM_RB_STATUS_i      => readback_active         -- : in
                std_logic

    );


end rtl;
```

L. X1 SHIFT REGISTER CONSTRAINT FILE (VIRTEX I)

(After [17])

```
# Pin assignments for X1
#
# double-check all pin assignments???
# these numbers derived from a Mar 2004 diagram
#

# clock is constrained at bottom of this file!!!
# system clock, pin 199 on X1
#NET "CLOCK" LOC = "P199";
#NET "CLOCK" PERIOD = 40;
# NET "s_clock" PERIOD = 80;
# the line above gave errors during "Translate"???

# signals to/from X2
NET "T_BITOUT_TO_X2_o" LOC = "p153";  # X1_X2_AUX<0>
NET "T_CE_TO_X2_o" LOC = "p151";  # X1_X2_AUX<1>
NET "T_RESET_TO_X2_o" LOC = "p150";  # X1_X2_AUX<2>
# NET "XXX" LOC = "p149";  # X1_X2_AUX<3>
# NET "XXX" LOC = "p147";  # X1_X2_AUX<4>
# NET "XXX" LOC = "p146";  # X1_X2_AUX<5>
# NET "XXX" LOC = "p145";  # X1_X2_AUX<6>
# NET "XXX" LOC = "p144";  # X1_X2_AUX<7>
# NET "XXX" LOC = "p135";  # X1_X2_AUX<8>
# NET "XXX" LOC = "p134";  # X1_X2_AUX<9>
NET "T_SUMDIFF_FROM_X2_i<0>" LOC = "p132";  # X1_X2_AUX<10>
NET "T_DOUT_FROM_X2_i" LOC = "p127";  # X1_X2_AUX<11>
NET "T_SUMDIFF_FROM_X2_i<1>" LOC = "p126";  # X1_X2_AUX<12>
NET "T_XOR_DOUT_FROM_X2_i" LOC = "p120";  # X1_X2_AUX<13>
NET "T_SUMDIFF_FROM_X2_i<2>" LOC = "p119";  # X1_X2_AUX<14>
#NET "T_DOUT_FROM_X2_i<2>" LOC = "p112";  # X1_X2_AUX<15>
NET "T_SUMDIFF_FROM_X2_i<3>" LOC = "p111";  # X1_X2_AUX<16>
#NET "T_DOUT_FROM_X2_i<3>" LOC = "p110";  # X1_X2_AUX<17>
NET "T_SUMDIFF_FROM_X2_i<4>" LOC = "p109";  # X1_X2_AUX<18>
#NET "T_DOUT_FROM_X2_i<4>" LOC = "p108";  # X1_X2_AUX<19>
NET "T_SUMDIFF_FROM_X2_i<5>" LOC = "p107";  # X1_X2_AUX<20>
#NET "T_DOUT_FROM_X2_i<5>" LOC = "p105";  # X1_X2_AUX<21>
NET "T_SUMDIFF_FROM_X2_i<6>" LOC = "p104";  # X1_X2_AUX<22>
#NET "T_DOUT_FROM_X2_i<6>" LOC = "p103";  # X1_X2_AUX<23>
NET "T_SUMDIFF_FROM_X2_i<7>" LOC = "p102";  # X1_X2_AUX<24>
#NET "T_DOUT_FROM_X2_i<7>" LOC = "p101";  # X1_X2_AUX<25>
NET "T_SUMDIFF_FROM_X2_i<8>" LOC = "p98";  # X1_X2_AUX<26>
#NET "T_DOUT_FROM_X2_i<8>" LOC = "p97";  # X1_X2_AUX<27>
NET "T_SUMDIFF_FROM_X2_i<9>" LOC = "p96";  # X1_X2_AUX<28>
#NET "T_DOUT_FROM_X2_i<9>" LOC = "p94";  # X1_X2_AUX<29>
NET "T_SUMDIFF_FROM_X2_i<10>" LOC = "p93";  # X1_X2_AUX<30>
#NET "T_DOUT_FROM_X2_i<10>" LOC = "p92";  # X1_X2_AUX<31>
NET "T_SUMDIFF_FROM_X2_i<11>" LOC = "p91";  # X1_X2_AUX<32>
#NET "T_DOUT_FROM_X2_i<11>" LOC = "p90";  # X1_X2_AUX<33>
NET "T_SUMDIFF_FROM_X2_i<12>" LOC = "p89";  # X1_X2_AUX<34>
#NET "T_DOUT_FROM_X2_i<12>" LOC = "p88";  # X1_X2_AUX<35>
NET "T_SUMDIFF_FROM_X2_i<13>" LOC = "p82";  # X1_X2_AUX<36>
#NET "T_DOUT_FROM_X2_i<13>" LOC = "p81";  # X1_X2_AUX<37>
NET "T_SUMDIFF_FROM_X2_i<14>" LOC = "p80";  # X1_X2_AUX<38>
```

```
#NET "T_DOUT_FROM_X2_i<14>" LOC = "p79";  # X1_X2_AUX<39>
NET "T_SUMDIFF_FROM_X2_i<15>" LOC = "p78";  # X1_X2_AUX<40>
#NET "T_DOUT_FROM_X2_i<15>" LOC = "p77";  # X1_X2_AUX<41>
# NET "XXX" LOC = "p75";  # X1_X2_AUX<42>
# NET "XXX" LOC = "p74";  # X1_X2_AUX<43>
# NET "XXX" LOC = "p71";  # X1_X2_AUX<44>

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
#Flash Interface Signals
NET "T_Flash_data_i<0>" LOC = "P207";
NET "T_Flash_data_i<1>" LOC = "P209";
NET "T_Flash_data_i<2>" LOC = "P212";
NET "T_Flash_data_i<3>" LOC = "P216";
NET "T_Flash_data_i<4>" LOC = "P218";
NET "T_Flash_data_i<5>" LOC = "P220";
NET "T_Flash_data_i<6>" LOC = "P223";
NET "T_Flash_data_i<7>" LOC = "P226";
NET "T_Flash_data_i<8>" LOC = "P208";
NET "T_Flash_data_i<9>" LOC = "P211";
NET "T_Flash_data_i<10>" LOC = "P213";
NET "T_Flash_data_i<11>" LOC = "P217";
NET "T_Flash_data_i<12>" LOC = "P219";
NET "T_Flash_data_i<13>" LOC = "P222";
NET "T_Flash_data_i<14>" LOC = "P224";
NET "T_Flash_data_i<15>" LOC = "P225";
NET "T_Flash_address_o<0>" LOC = "P206";
NET "T_Flash_address_o<1>" LOC = "P205";
NET "T_Flash_address_o<2>" LOC = "P204";
NET "T_Flash_address_o<3>" LOC = "P198";
NET "T_Flash_address_o<4>" LOC = "P197";
NET "T_Flash_address_o<5>" LOC = "P196";
NET "T_Flash_address_o<6>" LOC = "P195";
NET "T_Flash_address_o<7>" LOC = "P194";
NET "T_Flash_address_o<8>" LOC = "P182";
NET "T_Flash_address_o<9>" LOC = "P183";
NET "T_Flash_address_o<10>" LOC = "P184";
NET "T_Flash_address_o<11>" LOC = "P185";
NET "T_Flash_address_o<12>" LOC = "P188";
NET "T_Flash_address_o<13>" LOC = "P189";
NET "T_Flash_address_o<14>" LOC = "P190";
NET "T_Flash_address_o<15>" LOC = "P192";
NET "T_Flash_address_o<16>" LOC = "P193";
NET "T_Flash_address_o<17>" LOC = "P177";
NET "T_Flash_address_o<18>" LOC = "P178";
NET "T_Flash_address_o<19>" LOC = "P179";
NET "T_Flash_address_o<20>" LOC = "P181";
NET "T_Flash_WE_o" LOC = "P165";
NET "T_Flash_RP_o" LOC = "P166";
NET "T_Flash_WP_o" LOC = "P167";
NET "T_Flash_CE_A_o" LOC = "P164";
NET "T_Flash_CE_B_o" LOC = "P125"; # doesn't do anything!
NET "T_Flash_OE_o" LOC = "P162";

#PC/104 Interface Signals
NET "T_Data_io<0>" LOC = "P11";                  #ISA Data Bit 0 p. 11
NET "T_Data_io<1>" LOC = "P10";         #ISA Data Bit 1 p. 10
NET "T_Data_io<2>" LOC = "P9";          #ISA Data Bit 2 p. 09
NET "T_Data_io<3>" LOC = "P7";          #ISA Data Bit 3 p. 07
NET "T_Data_io<4>" LOC = "P6";          #ISA Data Bit 4 p. 06
NET "T_Data_io<5>" LOC = "P5";              #ISA Data Bit 5 p. 05
NET "T_Data_io<6>" LOC = "P4";              #ISA Data Bit 6 p. 04
NET "T_Data_io<7>" LOC = "P3";              #ISA Data Bit 7 p. 03
NET "T_Address_i<0>" LOC = "P47";       #ISA Address 0      p. 47
NET "T_Address_i<1>" LOC = "P46";       #ISA Address 1      p. 46
NET "T_Address_i<2>" LOC = "P45";       #ISA Address 2      p. 45
NET "T_Address_i<3>" LOC = "P39";       #ISA Address 3      p. 39
NET "T_Address_i<4>" LOC = "P36";       #ISA Address 4      p. 36
NET "T_Address_i<5>" LOC = "P34";       #ISA Address 5      p. 34
NET "T_Address_i<6>" LOC = "P32";       #ISA Address 6      p. 32
NET "T_Address_i<7>" LOC = "P29";       #ISA Address 7      p. 29
NET "T_Address_i<8>" LOC = "P25";       #ISA Address 8      p. 25
NET "T_Address_i<9>" LOC = "P23";       #ISA Address 9      p. 23
NET "T_IORead_i" LOC = "P19";                   #IO Read        p. 19
NET "T_IOWrite_i" LOC = "P17";              #IO Write       p. 17
NET "T_AEN_i" LOC = "P13";                      #Address Enable p. 13

# Selectmap interface signals
NET "T_CCLK_o" LOC = "P69";                     #Drive X2's CCLK pin
NET "T_SELECTMAP_INIT_o" LOC = "P117";          #Drive X2's INIT pin
NET "T_SELECTMAP_WRITE_o" LOC = "P176";         #Drive X2's WRITE pin
NET "T_SELECTMAP_CS_o" LOC = "P175";            #Drive X2's CS pin

# MLS swap pins so D(0) is LSB
NET "T_SELECTMAP_DATA_io<7>" LOC = "P169";
NET "T_SELECTMAP_DATA_io<6>" LOC = "P128";
NET "T_SELECTMAP_DATA_io<5>" LOC = "P131";
NET "T_SELECTMAP_DATA_io<4>" LOC = "P137";
NET "T_SELECTMAP_DATA_io<3>" LOC = "P148";
NET "T_SELECTMAP_DATA_io<2>" LOC = "P155";
NET "T_SELECTMAP_DATA_io<1>" LOC = "P158";
NET "T_SELECTMAP_DATA_io<0>" LOC = "P168";
#NET "SELECTMAP_BUSY_i" LOC = "P118";

NET "T_clock_i" LOC = "P199";
NET "T_x2_mode<0>" LOC = "P160";
NET "T_x2_mode<1>" LOC = "P159";
NET "T_x2_mode<2>" LOC = "P161";
NET "t_clock_i" PERIOD = 20;
NET "s_clock" PERIOD = 40;
NET "s_clock_x2" PERIOD = 80;

#NET "T_SELECTMAP_DATA_io<7>" TNM="smpins";
#NET "T_SELECTMAP_DATA_io<6>" TNM="smpins";
#NET "T_SELECTMAP_DATA_io<5>" TNM="smpins";
#NET "T_SELECTMAP_DATA_io<4>" TNM="smpins";
#NET "T_SELECTMAP_DATA_io<3>" TNM="smpins";
#NET "T_SELECTMAP_DATA_io<2>" TNM="smpins";
#NET "T_SELECTMAP_DATA_io<1>" TNM="smpins";
#NET "T_SELECTMAP_DATA_io<0>" TNM="smpins";
#NET "T_SELECTMAP_INIT_o" TNM="smpins";
#NET "T_SELECTMAP_WRITE_o" TNM="smpins";
#NET "T_SELECTMAP_CS_o" TNM="smpins";
```

```
#TIMESPEC "TS_1" = FROM : FFS : TO : PADS : 5 ns;

#TIMEGRP "smpins" OFFSET=OUT 5 AFTER "s_clock";
#TIMEGRP "smpins" OFFSET=IN 5 BEFORE "s_clock";


# "from_x2..." and "busy_out" and "SELECTMAP_BUSY_i" have been
                commented out!!!
#The X1_X2_AUX pins are defined at top of file!!!
#NET "busy_out" LOC = "P49";    #io pin 0
#NET "output_from_x2_8" LOC = "P64";    #io pin 10
#NET "TP1" LOC = "P70";       #io pin 15
#NET "TP1" FAST ;
#NET "TP2" LOC = "P65";      #io pin 11
#NET "TP2" FAST ;
#NET "TP3" LOC = "P64";      #io pin 10
#NET "TP3" FAST ;
```

M. X1 SHIFT REGISTER CONSTRAINT FILE (VIRTEX II)

(After [17])

```
# Pin checks, X1 Virtex1 Part on CFTP1 vs CFTP2:

# PC/104:
# DATA(7 downto 0) same
# ADDR(9 downto 0) same
# IOW, IOR, AEN same

# FLASH:
# DATA(15 downto 0) same
# ADDR(20 downto 0) same
# WE, WP, RP, CE_A, CE_B, OE same
# NOTE: Added second flash to CFTP2. FLASH_CE_B added
#       to P125. Just have this pin on both boards,
#       P125 isn't connected to anything on CFTP1
# CFTP2 board has VPPEN on pin 60!!!! need to pull low
#     in order to erase or write flash!!!
#     new CFTP1 board will have similar pin, old CFTP1
#     does not (EN always pulled active on Flash)
# CFTP2 board also has a different Flash device,
#     auto-locking blocks. schem says both CFTP1 and CFTP2
#     have C3 device, but CFTP1 actually has B3 device.
#     CFTP2 device is top boot, CFTP1 device is bottom boot

# X1/X2 AUX:
# double checked pin numbers, looks ok
# all pins are the same between CFTP1 and CFTP2 board,
# *****except X1/X2 Aux 42 is P75 on CFTP1 and P125 on CFTP2
# *****MLS 20051026: X1/X2 Aux 42 (P125) has been tied to CE pin
#       for Flash_B!!  DO NOT USE AS AUX PIN!!!
#    CFTP2 has no X1/X2 Aux 43 or 44 (P74 and P51 on
#    CFTP1, respectively)

# SELECTMAP:
# Drive X2's CCLK:
#   on old CFTP1: jumpered P69 to X2's CCLK
#   on CFTP2: Dedicated User I/O on X1 (P65)
#       connected to X2's CCLK
# Drive X2's INIT, WRITE, and CS pins
#   on old CFTP1: use X1's INIT(P117), WR(P176), CS(P175)
#       pins (turn user I/O after config)
#   on CFTP2: use dedicated I/O (P124, P63, P64, respectively)
# Drive X2's Data lines
#   on old CFTP1: connect D0 on X1 to D7 on X2, D1 to D6, etc
#       (turn user io after config)
#       In ucf file, I swap values because D7 is the LSB
#       in Selectmap mode, and I want D0 to be the LSB,
#       so I set it up so my signal SELECTMAP_DATA_io(0) is
#       connected to the actually X2_DATA(7) pin, etc
#   on CFTP2: dedicated user io on X1 connected to X2's pins
#       same thing with MSB/LSB swap
#       D0=68,D1=69,D2=70,D3=71,D4=74,D5=75,D6=121,D7=122

# MODE PINS:  SAME
# CLOCK:  SAME (oscillator on P199 on X1 FPGA)

# Pin assignments for X1, compatible with Virtex-II
#
# double-check all pin assignments???
# these numbers derived from a May 2005 diagram

# clock is constrained at bottom of this file!!!
# system clock oscillator, pin 199 on X1
# NET "T_CLOCK_i" LOC = "P199";
# NET "T_CLOCK_i" PERIOD = 40;
# NET "s_clock" PERIOD = 80;
# the line above gave errors during "Translate"???

# START signals to/from X2
NET "T_BITOUT_TO_X2_o" LOC = "p153";  # X1_X2_AUX<0>
NET "T_CE_TO_X2_o" LOC = "p151";  # X1_X2_AUX<1>
NET "T_RESET_TO_X2_o" LOC = "p150";  # X1_X2_AUX<2>
# NET "XXX" LOC = "p149";  # X1_X2_AUX<3>
# NET "XXX" LOC = "p147";  # X1_X2_AUX<4>
# NET "XXX" LOC = "p146";  # X1_X2_AUX<5>
# NET "XXX" LOC = "p145";  # X1_X2_AUX<6>
# NET "XXX" LOC = "p144";  # X1_X2_AUX<7>
# NET "XXX" LOC = "p135";  # X1_X2_AUX<8>
# NET "XXX" LOC = "p134";  # X1_X2_AUX<9>
NET "T_SUMDIFF_FROM_X2_i<0>" LOC = "p132";  # X1_X2_AUX<10>
NET "T_DOUT_FROM_X2_i" LOC = "p127";  # X1_X2_AUX<11>
NET "T_SUMDIFF_FROM_X2_i<1>" LOC = "p126";  # X1_X2_AUX<12>
NET "T_XOR_DOUT_FROM_X2_i" LOC = "p120";  # X1_X2_AUX<13>
NET "T_SUMDIFF_FROM_X2_i<2>" LOC = "p119";  # X1_X2_AUX<14>
#NET "T_DOUT_FROM_X2_i<2>" LOC = "p112";  # X1_X2_AUX<15>
NET "T_SUMDIFF_FROM_X2_i<3>" LOC = "p111";  # X1_X2_AUX<16>
#NET "T_DOUT_FROM_X2_i<3>" LOC = "p110";  # X1_X2_AUX<17>
NET "T_SUMDIFF_FROM_X2_i<4>" LOC = "p109";  # X1_X2_AUX<18>
#NET "T_DOUT_FROM_X2_i<4>" LOC = "p108";  # X1_X2_AUX<19>
NET "T_SUMDIFF_FROM_X2_i<5>" LOC = "p107";  # X1_X2_AUX<20>
#NET "T_DOUT_FROM_X2_i<5>" LOC = "p105";  # X1_X2_AUX<21>
NET "T_SUMDIFF_FROM_X2_i<6>" LOC = "p104";  # X1_X2_AUX<22>
#NET "T_DOUT_FROM_X2_i<6>" LOC = "p103";  # X1_X2_AUX<23>
NET "T_SUMDIFF_FROM_X2_i<7>" LOC = "p102";  # X1_X2_AUX<24>
#NET "T_DOUT_FROM_X2_i<7>" LOC = "p101";  # X1_X2_AUX<25>
NET "T_SUMDIFF_FROM_X2_i<8>" LOC = "p98";  # X1_X2_AUX<26>
#NET "T_DOUT_FROM_X2_i<8>" LOC = "p97";  # X1_X2_AUX<27>
NET "T_SUMDIFF_FROM_X2_i<9>" LOC = "p96";  # X1_X2_AUX<28>
#NET "T_DOUT_FROM_X2_i<9>" LOC = "p94";  # X1_X2_AUX<29>
NET "T_SUMDIFF_FROM_X2_i<10>" LOC = "p93";  # X1_X2_AUX<30>
#NET "T_DOUT_FROM_X2_i<10>" LOC = "p92";  # X1_X2_AUX<31>
NET "T_SUMDIFF_FROM_X2_i<11>" LOC = "p91";  # X1_X2_AUX<32>
#NET "T_DOUT_FROM_X2_i<11>" LOC = "p90";  # X1_X2_AUX<33>
NET "T_SUMDIFF_FROM_X2_i<12>" LOC = "p89";  # X1_X2_AUX<34>
#NET "T_DOUT_FROM_X2_i<12>" LOC = "p88";  # X1_X2_AUX<35>
NET "T_SUMDIFF_FROM_X2_i<13>" LOC = "p82";  # X1_X2_AUX<36>
#NET "T_DOUT_FROM_X2_i<13>" LOC = "p81";  # X1_X2_AUX<37>
NET "T_SUMDIFF_FROM_X2_i<14>" LOC = "p80";  # X1_X2_AUX<38>
#NET "T_DOUT_FROM_X2_i<14>" LOC = "p79";  # X1_X2_AUX<39>
NET "T_SUMDIFF_FROM_X2_i<15>" LOC = "p78";  # X1_X2_AUX<40>
```

```
#NET "T_DOUT_FROM_X2_i<15>" LOC = "p77";  # X1_X2_AUX<41>
# END signals to/from X2
#NET "CE for FLASH B on CFTP2 Board!!!!" LOC = "p125";  # DO NOT USE!!!
# NET "XXX" LOC = "XXX";  # X1_X2_AUX<43> not on Virtex-II board
# NET "XXX" LOC = "XXX";  # X1_X2_AUX<44> not on Virtex-II board

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
#Flash Interface Signals
NET "T_Flash_data_i<0>" LOC = "P207";
NET "T_Flash_data_i<1>" LOC = "P209";
NET "T_Flash_data_i<2>" LOC = "P212";
NET "T_Flash_data_i<3>" LOC = "P216";
NET "T_Flash_data_i<4>" LOC = "P218";
NET "T_Flash_data_i<5>" LOC = "P220";
NET "T_Flash_data_i<6>" LOC = "P223";
NET "T_Flash_data_i<7>" LOC = "P226";
NET "T_Flash_data_i<8>" LOC = "P208";
NET "T_Flash_data_i<9>" LOC = "P211";
NET "T_Flash_data_i<10>" LOC = "P213";
NET "T_Flash_data_i<11>" LOC = "P217";
NET "T_Flash_data_i<12>" LOC = "P219";
NET "T_Flash_data_i<13>" LOC = "P222";
NET "T_Flash_data_i<14>" LOC = "P224";
NET "T_Flash_data_i<15>" LOC = "P225";
NET "T_Flash_address_o<0>" LOC = "P206";
NET "T_Flash_address_o<1>" LOC = "P205";
NET "T_Flash_address_o<2>" LOC = "P204";
NET "T_Flash_address_o<3>" LOC = "P198";
NET "T_Flash_address_o<4>" LOC = "P197";
NET "T_Flash_address_o<5>" LOC = "P196";
NET "T_Flash_address_o<6>" LOC = "P195";
NET "T_Flash_address_o<7>" LOC = "P194";
NET "T_Flash_address_o<8>" LOC = "P182";
NET "T_Flash_address_o<9>" LOC = "P183";
NET "T_Flash_address_o<10>" LOC = "P184";
NET "T_Flash_address_o<11>" LOC = "P185";
NET "T_Flash_address_o<12>" LOC = "P188";
NET "T_Flash_address_o<13>" LOC = "P189";
NET "T_Flash_address_o<14>" LOC = "P190";
NET "T_Flash_address_o<15>" LOC = "P192";
NET "T_Flash_address_o<16>" LOC = "P193";
NET "T_Flash_address_o<17>" LOC = "P177";
NET "T_Flash_address_o<18>" LOC = "P178";
NET "T_Flash_address_o<19>" LOC = "P179";
NET "T_Flash_address_o<20>" LOC = "P181";
NET "T_Flash_WE_o" LOC = "P165";
NET "T_Flash_RP_o" LOC = "P166";
NET "T_Flash_WP_o" LOC = "P167";
NET "T_Flash_CE_A_o" LOC = "P164";
NET "T_Flash_CE_B_o" LOC = "P125";
NET "T_Flash_OE_o" LOC = "P162";

#PC/104 Interface Signals
NET "T_Data_io<0>" LOC = "P11";                 #ISA Data Bit 0 p. 11
NET "T_Data_io<1>" LOC = "P10";         #ISA Data Bit 1 p. 10
NET "T_Data_io<2>" LOC = "P9";          #ISA Data Bit 2 p. 09
NET "T_Data_io<3>" LOC = "P7";          #ISA Data Bit 3 p. 07
NET "T_Data_io<4>" LOC = "P6";          #ISA Data Bit 4 p. 06
NET "T_Data_io<5>" LOC = "P5";              #ISA Data Bit 5 p. 05
NET "T_Data_io<6>" LOC = "P4";              #ISA Data Bit 6 p. 04
NET "T_Data_io<7>" LOC = "P3";              #ISA Data Bit 7 p. 03
NET "T_Address_i<0>" LOC = "P47";       #ISA Address 0      p. 47
NET "T_Address_i<1>" LOC = "P46";       #ISA Address 1      p. 46
NET "T_Address_i<2>" LOC = "P45";       #ISA Address 2      p. 45
NET "T_Address_i<3>" LOC = "P39";       #ISA Address 3      p. 39
NET "T_Address_i<4>" LOC = "P36";       #ISA Address 4      p. 36
NET "T_Address_i<5>" LOC = "P34";       #ISA Address 5      p. 34
NET "T_Address_i<6>" LOC = "P32";       #ISA Address 6      p. 32
NET "T_Address_i<7>" LOC = "P29";       #ISA Address 7      p. 29
NET "T_Address_i<8>" LOC = "P25";       #ISA Address 8      p. 25
NET "T_Address_i<9>" LOC = "P23";       #ISA Address 9      p. 23
NET "T_IORead_i" LOC = "P19";                   #IO Read        p. 19
NET "T_IOWrite_i" LOC = "P17";              #IO Write       p. 17
NET "T_AEN_i" LOC = "P13";                      #Address Enable p. 13

# Selectmap interface signals
NET "T_CCLK_o" LOC = "P65"; #Drive X2's CCLK pin, CFTP2 ok MLS
NET "T_SELECTMAP_INIT_o" LOC = "P124"; #Drive X2's INIT pin, CFTP2 ok
NET "T_SELECTMAP_WRITE_o" LOC = "P63"; #Drive X2's WRITE pin, CFTP2 ok
NET "T_SELECTMAP_CS_o" LOC = "P64"; #Drive X2's CS pin, CFTP2 ok MLS

# MLS swap pins so D(0) is LSB
NET "T_SELECTMAP_DATA_io<7>" LOC = "P68"; #Drive X2's D0, CFTP2 ok MLS
NET "T_SELECTMAP_DATA_io<6>" LOC = "P69"; #Drive X2's D1, CFTP2 ok MLS
NET "T_SELECTMAP_DATA_io<5>" LOC = "P70"; #Drive X2's D2, CFTP2 ok MLS
NET "T_SELECTMAP_DATA_io<4>" LOC = "P71"; #Drive X2's D3, CFTP2 ok MLS
NET "T_SELECTMAP_DATA_io<3>" LOC = "P74"; #Drive X2's D4, CFTP2 ok MLS
NET "T_SELECTMAP_DATA_io<2>" LOC = "P75"; #Drive X2's D5, CFTP2 ok MLS
NET "T_SELECTMAP_DATA_io<1>" LOC = "P121"; #Drive X2's D6, CFTP2 ok MLS
NET "T_SELECTMAP_DATA_io<0>" LOC = "P122"; #Drive X2's D7, CFTP2 ok MLS

#NET "SELECTMAP_BUSY_i" LOC = "P118";
NET "T_clock_i" LOC = "P199";
NET "T_x2_mode<0>" LOC = "P160";
NET "T_x2_mode<1>" LOC = "P159";
NET "T_x2_mode<2>" LOC = "P161";
NET "t_clock_i" PERIOD = 20;
NET "s_clock" PERIOD = 40;
NET "s_clock_X2" PERIOD = 80;

# "busy_out" and "SELECTMAP_BUSY_i" have been commented out!!!

#The X1_X2_AUX pins are defined at top of file!!!
#NET "busy_out" LOC = "P49";    #io pin 0
#NET "output_from_x2_8" LOC = "P64";    #io pin 10
#NET "TP1" LOC = "P70";       #io pin 15
#NET "TP1" FAST ;
#NET "TP2" LOC = "P65";      #io pin 11
#NET "TP2" FAST ;
#NET "TP3" LOC = "P64";      #io pin 10
#NET "TP3" FAST ;
```

N. SHIFT REGISTER MODULE WITH SRL16E MACRO AND FLIP-FLOPS

```
-------------------------------------------------
-- filename: sr_testing.vhd
-- author: James Coudeyras (2005)
--
-- This file is for the SRL16E macro plus flip-flop (SRL+1)
-- used for X2 which will be used as the initial test
-- for proton radiation testing at UC-Davis.
--
-------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sr_testing is

    generic ( WIDTH : integer := 2040); -- should yield 120 SRL16/FF
                pairs
                -- SR length (2400 for C1, 300 for C1-no SRL, , 12000
                for C2)

    Port  ( clock       : in std_logic;
        reset           : in std_logic;
        ce              : in std_logic;
        din             : in std_logic;
        dout            : out std_logic;
        sumdiff         : out std_logic );

end sr_testing;

architecture sr_sequence of sr_testing is
    signal reg_a    : std_logic_vector (WIDTH-1 downto 0);
    signal reg_b    : std_logic_vector (WIDTH-1 downto 0);

begin

process (clock, reset)
begin
    if (reset = '1') then

        sumdiff <= '0';
        dout <=  '0';

    elsif (clock'event and clock='1') then

        sumdiff <= '0';

        if ce='1' then
            reg_a <= din & reg_a(WIDTH-1 downto 1);
            reg_b <= din & reg_b(WIDTH-1 downto 1);
        end if;
-- LABEL1:
        for I in 1 to (WIDTH/17) loop
            if ( (reg_a((I-1)*17) xor reg_b((I-1)*17)) = '1' ) or
```

```
                                        ((reg_a(((I-1)*17)+1) xor
                reg_b(((I-1)*17)+1)) = '1' ) then
                sumdiff <= '1';
            end if;
        end loop;

        dout <= reg_a(0);

    end if;


end process;

end sr_sequence;
```

## O.	SHIFT REGISTER MODULE FLIP-FLOPS ONLY

```
-------------------------------------------------
-- filename: sr_testing.vhd
-- author: James Coudeyras (2005)
--
-- This file is for the flip-flop only shift register (noSRL)
-- used for X2 which will be used as the initial test
-- for proton radiation testing at UC-Davis.
--
-------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sr_testing is

    generic ( WIDTH : integer := 320);
                -- SR length (2400 for C1, 300 for C1-no SRL, , 12000
                for C2)

    Port  ( clock       : in std_logic;
        reset           : in std_logic;
        ce              : in std_logic;
        din             : in std_logic;
        dout            : out std_logic;
        sumdiff         : out std_logic );

end sr_testing;

architecture sr_sequence of sr_testing is
    signal reg_a    : std_logic_vector (WIDTH-1 downto 0);
    signal reg_b    : std_logic_vector (WIDTH-1 downto 0);

begin

process (clock, reset)
begin
    if (reset = '1') then

        sumdiff <= '0';
        dout <=  '0';

    elsif (clock'event and clock='1') then

        sumdiff <= '0';

        if ce='1' then
            reg_a <= din & reg_a(WIDTH-1 downto 1);
            reg_b <= din & reg_b(WIDTH-1 downto 1);
        end if;
-- LABEL1:
        for I in 1 to (WIDTH/16) loop
            if ( (reg_a((I-1)*16) xor reg_b((I-1)*16)) = '1' ) then
                sumdiff <= '1';
            end if;
        end loop;

        dout <= reg_a(0);

    end if;

end process;

end sr_sequence;
```

P. MATLAB SCRIPT FOR DATA ANALYSIS

```
% Author: Josh Snodgrass, PhD Candidate, Naval Postgraduate School
% Nov 2005
% Reads ASCII log file from CFTP testing and provides SM Readback stats
% Modified: James Coudeyras, EE Master's Student, NPS
% Added SEU count and rate.

clear all; close all; clc;

fname='sr_SRLp1_c1_run2.log';
fdescrip = 'SRL & FF, Virtex 1, run 2';

fid=fopen(fname,'r');
if (fid==-1);   error('Invalid filename');  end;
frewind(fid);

SMold=0;
SMnew=0;
SMstart=0;
SMnewtrend=[];
zero2one=0;
one2zero=0;
NumRecon=0;

logline=[];

while (~feof(fid))
  logline=fgetl(fid);

  if (length(logline)>18)
    if (logline(1:18)=='Selectmap Reconfig')
      SMold=0;
      SMarray=[];
      NumRecon = NumRecon + 1;
    elseif (logline(1:18)=='Selectmap Readback')
      SMstart=1;
      SMnew=0;
    elseif (logline(1:5)=='Read:')
      NewEntry=1;  % assume we have a new entry
      for (i=1:1:SMold+SMnew)
        if (logline(1:50)==SMarray(i,1:50))  % all lines have > 50
characters
          NewEntry=0;  % negate flag if entry already exists
        end;
      end;
      if (NewEntry)
        SMnew=SMnew+1;
        SMarray(SMold+SMnew,1:length(logline))=logline;
        %disp(logline);  % show each of the new SEUs
        badbyte=dec2bin2(hex2dec(logline(7:8)));
        goodbyte=dec2bin2(hex2dec(logline(21:22)));
        zero2one=zero2one+sum((badbyte-goodbyte)>0);
        one2zero=one2zero+sum((goodbyte-badbyte)>0);
        if (abs(sum(xor(badbyte,goodbyte)))>1)
            disp(logline),
        end
      end
    end
  elseif strcmp(logline,'')&(SMstart)
    fprintf('Selectmap Readback #%i\n',length(SMnewtrend)+1);
    fprintf('Total SEUs = Old SEUs + New SEUs\n');
    fprintf('      %-4i =     %-4i +     %-
4i\n',SMold+SMnew,SMold,SMnew);
    fprintf('# of 0->1 = %i,  # of 1->0 = %i\n\n',zero2one,one2zero);
    SMold=SMold+SMnew;
    SMnewtrend(length(SMnewtrend)+1)=SMnew;
    SMstart=0;
  end
end

fclose('all');

figure(1)
stem(SMnewtrend,'b');
title(['SEUs Detected via Selectmap Readback, design: ' fdescrip]);
xlabel('Selectmap Readback Index @ 30 sec interval');
ylabel('SEUs per interval');
text(10,6,'Lower Flux')

NEW_ERR = []; m = 0;
NEW_RERR_ind = find(SMnewtrend);
for i=1:length(NEW_RERR_ind)
    m = NEW_RERR_ind(i);
    NEW_RERR(i) = SMnewtrend(m);
end
NEW_RERR_tot = sum(NEW_RERR)
NEW_RERR_ave = NEW_RERR_tot/length(NEW_RERR_ind)

RECON_ERR = zeros(1,length(SMnewtrend));
k = 1;
for i = 1:length(SMnewtrend)
    if (SMnewtrend(i) >= 1)
        RECON_ERR(k) = RECON_ERR(k) + SMnewtrend(i);
    else
        k = k + 1;
    end
end
RECONlength_ind = find(RECON_ERR);
RECONlength = max(RECONlength_ind);
%RECONstart = min(RECONlength_ind);
RECON_ERR = RECON_ERR(1:RECONlength);
NumRecon
ERR_btwn_RECON_ave = sum(RECON_ERR)/NumRecon

figure(2)
stem(RECON_ERR,'b');
title(['# of SEUs before Reconfiguration, design: ' fdescrip]);
xlabel('Reconfiguration'); ylabel('SEUs');
text(1.5,65,'Lower Flux')
```

LIST OF REFERENCES

1. "Moore's Law." 2005. Wikipedia, updated 29 November 2005. http://en.wikipedia.org/wiki/Moore's_law [29 November 2005].

2. Fuller, Earl, Michael Caffrey, Anthony Salazar, Carl Carmichael, Joe Fabula. 2000. "Radiation Characterization, and SEU Mitigation, of the Virtex FPGA for Space-Based Reconfigurable Computing." Proceedings of the 2nd Military and Aerospace Applications of Programmable Devices (MAPLD) and Technologies Conference Proceedings. September.

3. Guccione, Steve. Delon Levi, Prasanna. Sundararajan. 1999. "JBits: A Java-based Interface for Reconfigurable Computing." MAPLD 1999 Proceedings. September.

4. Lashomb, Peter A. 2002. Triple Modular Redundant (TMR) Microprocessor System for Field-Programmable Gate Array (FPGA) Implementation. Master's Thesis. Naval Postgraduate School.

5. Ebert, Dean A., Charles A. Hulme, Herschel H. Loomis, Alan A. Ross. 2003. "Configurable Fault Tolerant Processor (CFTP) for Space-Based Applications." 17th Annual AIAA/USU Conference on Small Satellites. SSC03-XI-5. August.

6. Snodgrass, Josh. 2005. Notes on CORDIC Algorithms. Naval Postgraduate School, Monterey.

7. Majewicz, Pete. 2005. Implementation of a Configurable Fault Tolerant Processor (CFTP) Using Internal Triple Modular Redundancy (TMR). Master's Thesis. Naval Postgraduate School.

8. Hulme, Charles A. 2003. Testing And Evaluation Of The Configurable Fault Tolerant Processor (CFTP) For Space-Based Applications. Master's Thesis. Naval Postgraduate School.

9. Wertz, James, and Wiley Larson. 2003. *Space Mission Analysis and Design*. El Segundo: Microcosm Press.

10. Fuller, Earl, Michael Caffrey, Anthony Salazar, Carl Carmichael, Joe Fabula. 2002. “Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex FPGA for Space Reconfigurable Computing.” MAPLD 2002 Proceedings. September.

11. Xilinx JBits 2.8 SDK for Virtex. 1999. Xilinx, Incorporated. San Jose, CA.

12. Rigmaiden, David. 2005. Interview by author, 31 October, Monterey. Transcript. Naval Postgraduate School, Monterey.

13. Loomis, Herschel and Alan Ross. 2005. “Configurable Fault Tolerant Processor for High-Radiation Orbit” Status Report to Space Systems Academic Group. 17 November. Naval Postgraduate School, Monterey.

14. Xilinx Integrated Software Environment (ISE) Ver. 6.3i. 2004. Xilinx, Incorporated. San Jose, CA.

15. Wu, C. Thomas. 2006. *An Introduction to Object-Oriented Programming with Java*. Boston: McGraw Hill.

16. Model Technology ModelSim SE 5.8a. 2003. Mentor Graphics. Wilsonville, OR.

17. Surratt, Mindy. 2005. *CFTP Development Environment Technical Manual*. Naval Postgraduate School, Monterey.

18. JTAG Programmer Guide. 2005. Xilinx, Incorporated. http://toolbox.xilinx.com/docsan/data/alliance/jtg/jtg.htm. [4 December 2005].

19. Toth, Frank. 2001. “iMPACT - New Configuration and Programming Software.” *XCell Journal Online*. Issue 41. http://www.xilinx.com/publications/products/software/xc_impact.htm. [30 November 2005].

20. “Configuring Virtex FPGAs from Parallel EPROMs with a CPLD.” 1999. Xilinx Application Note XAPP137. 1 March.

21. “Virtex FPGA Series Configuration and Readback.” 2005. Xilinx Application Note XAPP138. 11 March.

22. Andraka, Ray. 1998. "A survey of CORDIC algorithms for FPGA based computers." Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays. February.

23. Weatherford, Todd R. 2004. Notes on Space Qualification of Electronics. Naval Postgraduate School, Monterey.

24. Space Systems Academic Group. 2005. NPSAT1 System-Level Launch and Space Environments Test Plan. Monterey, CA: Naval Postgraduate School. Photocopied.

25. "Expanded View of NPSat-1." 2002. Naval Postgraduate School Space Systems Academic Group, updated 21 October 2003. http://intranet.nps.navy.mil/npsat/npsat1/default.htm [22 November 2005].

26. "Radiation Effects & Mitigation Overview." 2004. Xilinx Presentation. http://www.xilinx.com/esp/mil_aero/collateral/presentations/radiation_effects.pdf. [22 November 2005].

27. Mills, Carl S., Glenn Hines, Kim R. Fowler, M. Ann Garrison Darrin, Richard F. Conde, Harry A. C. Eaton. 2003. *Adaptive Data Analysis and Processing Technology (ADAPT) for Spacecraft*. Hampton: NASA Langley Research Center, Electronic Systems Brand, System Engineering Competency. ESTC 2003.

28. Johnson, D. Eric, Ketih S. Morgan, Michael J. Wirthlin, Michael P. Caffrey, Paul S. Graham. 2004. *Detection of Configuration Memory Upsets Causing Persistent Errors in SRAM-based FPGAs.* Los Alamos: Los Alamos National Laboratory. LA-UR-04-7085.

29. "PN Generators Using the SRLMacro." 2004. Xilinx Application Note XAPP211. 14 June.

30. "Linear Feedback Shift Registers in Virtex Devices." 2001. Xilinx Application Note XAPP210. 9 Jan.

31. MATLAB Ver. 7.1.0.246 (R14) Service Pack 3. 2005. Mathworks, Incorporated. Natick, MA.

32. Cosmic Ray Effects on MicroElectronics (CREME96). 1996. Naval Research Laboratory. Washington, DC.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Hersch Loomis
   Naval Postgraduate School
   Monterey, California

4. Professor Alan Ross
   Naval Postgraduate School
   Monterey, California